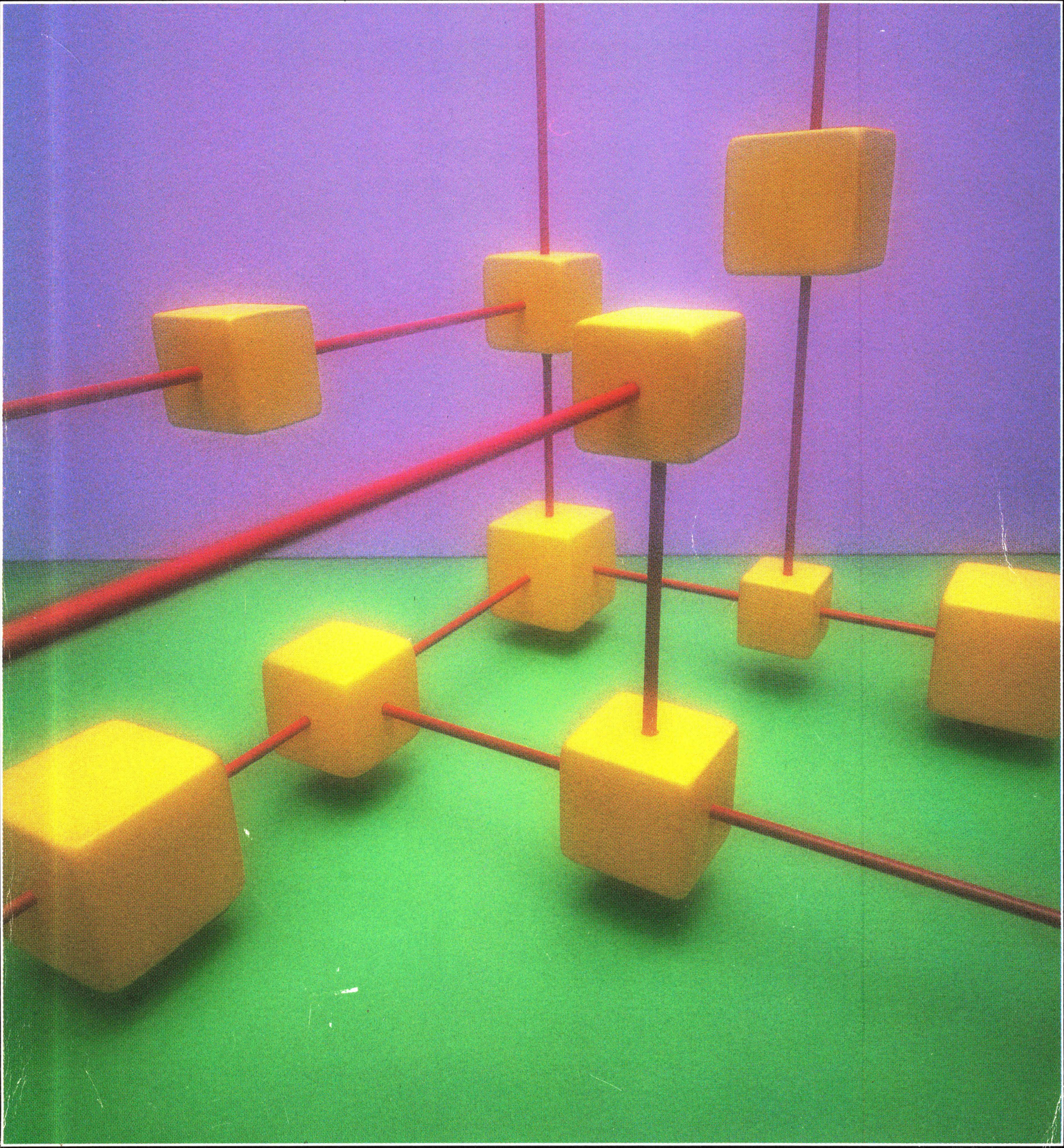
Microcommunications Handbook

Components, Boards, Systems, Software for Microcomputer Communications



Order Number: 231658-002

LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Literature:
(800) 548-4725

Use the order blank on the facing page or call our **Toll Free** Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for outside U.S. customers.

1987 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	*PRICE IN U.S. DOLLARS
COMPLETE SET OF 9 HANDBOOKS Save $50.00 off the retail price of $175.00	231003	**$125.00**
MEMORY COMPONENTS HANDBOOK	210830	**$18.00**
MICROCOMMUNICATIONS HANDBOOK	231658	**$20.00**
EMBEDDED CONTROLLER HANDBOOK (includes Microcontrollers and 8085, 80186, 80188)	210918	**$18.00**
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	**$25.00**
DEVELOPMENT TOOLS HANDBOOK	210940	**$18.00**
DOS DEVELOPMENT SOFTWARE CATALOG	280199	N/C
OEM BOARDS AND SYSTEMS HANDBOOK	280407	**$18.00**
MILITARY HANDBOOK	210461	**$18.00**
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997	**$20.00**
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	**$20.00**
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
LITERATURE PRICE LIST List of Intel Literature	210620	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C

*These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.

intel

SALES DESKTOP COPY 1-800-722-4726

032901-X

LITERATURE SALES ORDER FORM

1-800-842-3085

Mary Ann Sweeny 1-800-842-3085

NAME: ______________________________

COMPANY: ______________________________

ADDRESS: ______________________________

CITY: ______________________ STATE: __________ ZIP: __________

COUNTRY: ______________________________

PHONE NO.: (   ) ______________________________

ORDER NO.	TITLE	QTY.		PRICE		TOTAL
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	
			×		=	

Subtotal __________

Must Add Your Local Sales Tax __________

Must add appropriate postage to subtotal (10% U.S. and Canada, 20% all other) → Postage __________

Total __________

Pay by Visa, MasterCard, American Express, Check, Money Order, or company purchase order payable to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ Visa ☐ MasterCard ☐ American Express Expiration Date __________

Account No. ______________________________

Signature: ______________________________

Mail To: Intel Literature Sales
P.O. Box 58130
Santa Clara, CA
95052-8130

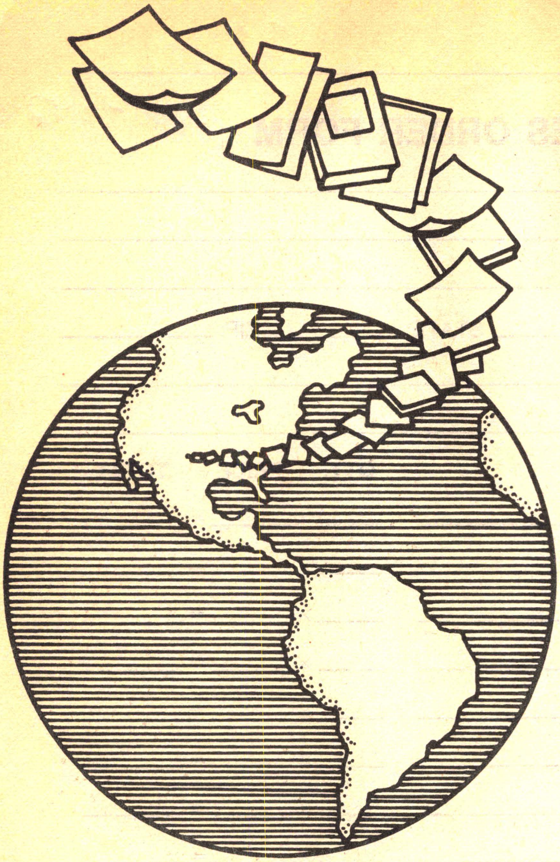
International Customers outside the U.S. and Canada should contact their local Intel Sales Office or Distributor listed in the back of most Intel literature.

European Literature Order Form in back of book.

Call Toll Free: **(800) 548-4725** for phone orders

Prices good until 12/31/87.

Source HB



We Bring Our World to Your Door

Intel's New Product Literature Subscription Service.

Keeping up with today's technology takes a lot of time and effort. With Intel's new Literature Subscription Service you will receive a package of current literature plus automatic quarterly updates on all the latest product and service news from Intel. From microprocessors — to peripherals and memories — to OEM boards, systems and software, you can choose to receive information from one, or all three, product categories for an entire year at a low one-time cost.

Save Time and Money. Subscribe Today.

Save 10% when ordering two or more packages.

To order, use the literature order form provided in this book or call TOLL FREE 800-548-4725

The charge for this service covers our printing, postage and handling costs only.

intel®

Each Literature Package Contains:

Newly published Data Sheets, Fact Sheets, Application Notes, Reliability Reports, Errata Reports, Article Reprints, Promotional Offers, Brochures, Flyers, Benchmark Reports, Technical Papers and more...

In Addition, Each Individual Package Contains:

1

Microprocessors, etc.

Product Line Handbooks on Microprocessors, Development Tools and Embedded Controllers.
plus
Quality/Reliability Information, The Product Guide, Literature Guide, Packaging Information, and other supporting information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail Value of Handbooks alone: $81)

Your price for the complete package with quarterly updates: $70

ORDER BY MARCH 31: Only $60
Order Number: 555100

2

Peripherals, Memories, etc.

Product Line Handbooks on Peripherals, Microcommunications, Memories and EPLDs.
plus
Quality/Reliability Information, The Product Guide, The Literature Guide, Packaging Information and other supporting information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail value of Handbooks alone: $83)

Your price for the complete package with quarterly updates: $70

ORDER BY MARCH 31: Only $60
Order Number: 555101

3

OEM Boards, Systems and Software

Product Line Handbooks on OEM Boards and Systems.
plus
Quality/Reliability Information, The Product Guide, The Literature Guide, and other supporting information.
plus
Three Quarterly Updates containing all new documentation on these products.
(Retail Value of Handbooks alone: $38)

Your price for the complete package with quarterly updates: $60

ORDER BY MARCH 31: Only $50
Order Number: 555102

Customers outside the U.S. and Canada should order directly from the U.S. on the U.S. literature order form.

Offer expires 12/31/87

intel®

MICROCOMMUNICATIONS HANDBOOK

1987

About Our Cover:

Intel provides the most comprehensive microsystems communications capabilities while supporting industry standards. The completeness of our solutions guarantees lower design risk and cost, better R&D leverage and a shorter time to market. Integrated communications and connectivity in multiple environments are the results of Intel's continued advances in VLSI technology. Networking software and communications modules provide the building blocks for the diverse communication needs in todays commercial and industrial environments. The image on our cover is a visual abstraction of this concept: the cubes representing microcomputer systems providing the communication capabilities at the heart of today's vast communication networks, be they, wide, local of small networks.

Concept/Design: Hall Kelley, Concept/Photography: R. J. Muna

intel

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, î, ICE, iCEL, iCS, iDBP, iDIS, I²ICE, iLBX, Inboard, i_m, iMDDX, iMMX, Insite, Intel, int_el, int_elBOS, Intelevision, int_eligent Identifier, int_eligent Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix 4-SITE.

Ethernet is a trademark of Xerox.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Literature Department
SC6-59
P.O. Box 58065
Santa Clara, CA 95052-8065

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Alphanumeric Index

Alphanumeric Index (Continued)

CUSTOMER SUPPORT

CUSTOMER SUPPORT

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, and consulting services. For more information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It includes factory repair services and worldwide field service offices providing hardware repair services, software support services, customer training classes, and consulting services.

HARDWARE SUPPORT SERVICES

Intel is committed to providing an international service support package through a wide variety of service offerings available from Intel Hardware Support.

SOFTWARE SUPPORT SERVICES

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and COMMENTS Magazine). Basic support includes updates and the subscription service. Contracts are sold in environments which represent product groupings (i.e., iRMX environment).

CONSULTING SERVICES

Intel provides field systems engineering services for any phase of your development or support effort. You can use our systems engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training, and customizing or tailoring an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, bitbus and LAN applications.

OVERVIEW

Imagine for a moment a world where all electronic communications were instantaneous. A world where voice, data, and graphics could all be transported via telephone lines to a variety of computers and receiving systems. A world where the touch of a finger could summon information ranging from stock reports to classical literature and bring it into environments as diverse as offices and labs, factories and living rooms.

Unfortunately, these promises of the Information Age still remain largely unfulfilled. While computer technology has accelerated rapidly over the last twenty years, the communications methods used to tie the wide variety of electronic systems in the world together have, by comparison, failed to keep pace. Faced with a tangle of proprietary offerings, high costs, evolving standards, and incomplete technologies, the world is still waiting for networks that are truly all-encompassing, the missing links to today's communications puzzle.

Enter microcommunications—microchip-based digital communications products and services. A migration of the key electronics communications functions into silicon is now taking place, providing the vital interfaces that have been lacking among the various networks now employed throughout the world. Through the evolution of VLSI (Very Large Scale Integration) technology, microcommunications now can offer the performance required to effect these communications interfaces at affordable costs, spanning the globe with silicon to eradicate the troublesome bottleneck that has plagued information transfer during recent years.

"There are three parts to the communications puzzle," says Gordon Moore, Intel Chairman and CEO. "The first incorporates the actual systems that communicate with each other, and the second is the physical means to connect them—such as cables, microwave technology, or fiber optics. It is the third area, the interfaces between the systems and the physical links, where silicon will act as the linchpin. That, in essence, is what microcommunications is all about."

THE COMMUNICATIONS BOTTLENECK

Visions of global networks are not new. Perhaps one of the most noteworthy of these has been espoused by Dr. Koji Kobayashi, chairman of NEC Corporation. His view of the future, developed over the nearly fifty years of his association with NEC, is known as C&C (Computers and Communications). It defines the marriage of passive communications systems and computers as processors and manipulators of information, providing the foundation for a discipline that is changing the basic character of modern society.

Kobayashi's macro vision hints at the obstacles confronting the future of C&C. When taken to the micro level, to silicon itself, one begins to understand the complexities that are involved. When Intel invented the microprocessor fifteen years ago, the first seeds of the personal computer revolution were sown , marking an era that over the last decade has dramatically influenced the way people work and live. PCs now proliferate in the office, in factories, and throughout laboratory environments. And their "intimidation" factor has lessened to where they are also becoming more and more prevalent in the home, beginning to penetrate a market that to date has remained relatively untapped.

Thanks to semiconductor technology, the personal computer has raised the level of productivity in our society. But most of that productivity has been gained by individuals at isolated workstations. Group productivity, meanwhile, still leaves much to be desired. The collective productivity of organizations can only be enhanced through more sophisticated networking technology. We are now faced with isolated "islands of automation" that must somehow be developed into networks of productivity.

But no amount of computing can meet these challenges if the corresponding communications technology is not sufficiently in step. The Information Age can only grow as fast as the lowest common denominator—which in this case is the aggregate communications bandwidth that continues to lag behind our increased computing power. Such is the nature of the communications bottleneck, where the growing amounts of information we are capable of generating can only flow as fast as the limited and incompatible communications capabilities now in place. Clearly, a crisis is at hand.

BREAKING UP THE BOTTLENECK

Three factors have contributed to this logjam: lack of industry standards, an insufficient cost/performance ratio, and the incomplete status of available communications technology to date.

- Standards—One look at the tangle of proprietary systems now populating office, factory, and laboratory environments gives a good indication of the inherent difficulty in hooking these diverse systems together. And these systems do not merely feature different architectures—they also represent completely different levels of computing, ranging from giant mainframes at one end of the scale down to individual microcontrollers on the other.

  The market has simply grown too fast to effectively accommodate the changes that have occurred. Suppliers face the dilemma of meshing product differentiation issues with industry-wide compatibility as

they develop their strategies; opting for one in the past often meant forsaking the other. And while some standards have coalesced, the industry still faces a technological Tower of Babel, with many proprietary solutions vying to be recognized in leadership positions.

- Cost/Performance Ratio—While various communications technologies struggle toward maturity, the industry has had to cope with tremendous costs associated with interconnectivity and interoperation. Before the shift to microelectronic interfaces began to occur, these connections often were prohibitively expensive.

  Says Ron Whittier, Intel Vice President and Director of Marketing: "Mainframes offer significant computing and communications power, but at a price that limits the number of users. What is needed is cost-effective communications solutions to hook together the roughly 16 million installed PCs in the market, as well as the soon-to-exist voice/data terminals. That's the role of microcommunications—bringing cost-effective communications solutions to the microcomputer world."

- Incomplete Technology—Different suppliers have developed many networking schemes, but virtually all have been fragmented and unable to meet the wide range of needs in the marketplace. Some of these approaches have only served to create additional problems, making OEMs and systems houses loathe to commit to suppliers who they fear cannot provide answers at all of the levels of communications that are now funneled into the bottleneck.

THE NETWORK TRINITY

Three principal types of networks now comprise the electronic communications marketplace: Wide Area Networks (WANs), Local Area Networks (LANs), and Small Area Networks (SANs). Each in its own fashion is turning to microcommunications for answers to its networking problems.

WANs—known by some as Global Area Networks (GANs)—are most commonly associated with the worldwide analog telephone system. The category also includes a number of other segments, such as satellite and microwave communications, traditional networks (like mainframe-to-mainframe connections), modems, statistical multiplexers, and front-end communications processors. The lion's share of nodes—electronic network connections—in the WAN arena, however, resides in the telecommunications segment. This is where the emerging ISDN (Integrated Services Digital Network) standard comes into focus as the most visible portion of the WAN marketplace.

The distances over which information may be transmitted via a WAN are essentially unlimited. The goal of ISDN is to take what is largely an analog global system and transform it into a digital network by defining the standard interfaces that will provide connections at each node.

These interfaces will allow basic digital communications to occur via the existing twisted pair of wires that comprise the telephone lines in place today. This would bypass the unfeasible alternative of installing completely new lines, which would be at cross purposes with the charter of ISDN: to reduce costs and boost performance through realization of an all-digital network.

The second category, Local Area Networks, represents the most talked-about link provided by microcommunications. In their most common form, LANs are comprised of—but not limited to—PC-to-PC connections. They incorporate information exchange over limited distances, usually not exceeding five kilometers, which often takes place within the same building or between adjacent work areas. The whole phenomenon surrounding LAN development, personal computing, and distributed processing essentially owes its existence to microcomputer technology, so it is not surprising that this segment of networking has garnered the attention it has in microelectronic circles.

Because of that, progress is being made in this area. The most prominent standard—which also applies to WANs and SANs—is the seven-layer Open Systems Interconnection (OSI) Model, established by the International Standards Organization (ISO). The model provides the foundation to which all LAN configurations must adhere if they hope to have any success in the marketplace. Interconnection protocols determining how systems are tied together are defined in the first five layers. Interoperation concepts are covered in the upper two layers, defining how systems can communicate with each other once they are tied together.

In the LAN marketplace, a large number of networking products and philosophies are available today, offering solutions at various price/performance points. Diverse approaches such as StarLAN, Token Bus and Token Ring, Ethernet, and PC-NET, to name a few of the more popular office LAN architectures, point to many choices for OEMs and end users.

A similar situation exists in the factory. While the Manufacturing Automation Protocol (MAP) standard is coalescing around the leadership of General Motors,

Boeing, and others, a variety of proprietary solutions also abound. The challenge is for a complete set of interfaces to emerge that can potentially tie all of these networks together in—and among—the office, factory, and lab environments.

The final third of the network trinity is the Small Area Network (SAN). This category is concerned with communications over very short distances, usually not exceeding 100 meters. SANs most often deal with chip-to-chip or chip-to-system transfer of information; they are optimized to deal with real-time applications generally managed by microcontrollers, such as those that take place on the factory floor among robots at various workstations.

SANs incorporate communications functions that are undertaken via serial backplanes in microelectronic equipment. While they represent a relatively small market in 1986 when compared to WANs and LANs, a tenfold increase is expected through 1990. SANs will have the greatest number of nodes among network applications by the next decade, thanks to their preponderance in many consumer products.

While factory applications will make up a large part of the SAN marketplace probably the greatest contributor to growth will be in automotive applications. Microcontrollers are now used in many dashboards to control a variety of engine tasks electronically, but they do not yet work together in organized and efficient networks. As Intel's Gordon Moore commented earlier this year to the New York Society of Security Analysts, when this technology shifts into full gear during the next decade, the total automobile electronics market will be larger than the entire semiconductor market was in 1985.

MARKET OPPORTUNITIES

Such growth is also mirrored in the projections for the WAN and LAN segments, which, when combined with SANs, make up the microcommunications market pie. According to Intel analysts, the total silicon microcommunications market in 1985 amounted to $522 million. By 1989, Intel predicts this figure will have expanded to $1290 million, representing a compounded annual growth rate of 25%.

And although the WAN market will continue to grow at a comfortable rate, the SAN and LAN pieces of the pie will increase the most dramatically. Whereas SANs represented only about 12.5% ($65 million) in 1985, they could explode to 22.5% ($290 million) of the larger pie by 1989. This growth is paralleled by increases in the LAN segment, which should grow from 34.5% of the total silicon microcommunications market in 1985 to 44.5% of the expanded pie in 1989.

Opportunities abound for microcommunications suppliers as the migration to silicon continues. And perhaps no VLSI supplier is as well-positioned in this marketplace as Intel, which predicts that 50% of its products will be microcommunications-related by 1990. The key here is the corporation's ability to bridge the three issues that contribute to the communications bottleneck: standards, cost-performance considerations, and the completeness of microcomputer and microcommunications product offerings.

INTEL AND VLSI: THE MICROCOMMUNICATIONS MATCH

Intel innovations helped make the microcomputer revolution possible. Such industry "firsts" include the microprocessor, the EPROM, the E2PROM, the microcontroller, development systems, and single board computers. Given this legacy, it is not surprising that the corporation should come to the microcommunications marketplace already equipped with a potent arsenal of tools and capabilities.

The first area centers on industry standards. As a VLSI microelectronic leader, Intel has been responsible for driving many of the standards that are accepted by the industry today. And when not actually initiating these standards, Intel has supported other existing and emerging standards through its longtime "open systems" philosophy. This approach protects substantial customer investments and ensures easy upgradability by observing compatibility with previous architectures and industry-leading standards.

Such a position is accentuated by Intel's technology relationships and alliances with many significant names in the microcommunications field. Giants like AT&T in the ISDN arena, General Motors in factory networking, and IBM in office automation all are working closely with Intel to further the standardization of the communications interfaces that are so vital to the world's networking future.

Cost/performance considerations also point to Intel's strengths. As a pioneer in VLSI technology, Intel has been at the forefront of achieving greater circuit densities and performance on single pieces of silicon: witness the 275,000 transistors housed on the 32-bit 80386, the highest performance commercial microprocessor ever built. As integration has increased, cost-per-bit has decreased steadily, marking a trend that remains consistent in the semiconductor industry. And one thing is

certain: microcommunications has a healthy appetite for transistors, placing it squarely in the center of the VLSI explosion.

But it is in the final area—completeness of technology and products—where Intel is perhaps the strongest. No other microelectronic vendor can point to as wide an array of products positioned across the various segments that comprise the microelectronic marketplace. Whether it be leadership in the WAN marketplace as the number one supplier of merchant telecommunications components, strength in SANs with world leadership in microcontrollers, or overall presence in the LAN arena with complete solutions in components, boards, software, and systems, Intel is a vital presence in the growing microcommunications arena.

That leadership extends beyond products. Along with its own application software, Intel is promoting expansion through partnerships with many different independent software vendors (ISVs), ensuring that the necessary application programs will be in place to fuel the gains provided by the silicon "engines" residing at the interface level. And finally, the corporation's commitment to technical support training, service, and its strong force of field applications engineers guarantees that it will back up its position and serve the needs that will continue to spring up as the microcommunications evolution becomes a reality.

Together, all the market segment alluded to in this article comprise the world of microcommunications, a world coming closer together every day as the web of networking solutions expands—all thanks to the technological ties that bind, reaching out to span the globe with silicon.

Local Area Networks

1

82586
LOCAL AREA NETWORK COPROCESSOR

- Performs Complete CSMA/CD Medium Access Control Functions Independently of CPU
  - High Level Command Interface
- Supports Established and Emerging LAN Standards
  - IEEE 802.3/Ethernet (10BASE5)
  - IEEE 802.3/Cheapernet (10BASE2)
  - IBM PC Network
  - IEEE 802.3/StarLAN (1BASE5)
  - Proprietary CSMA/CO Networks up to 10 Mbps
- On-Chip Memory Management
  - Automatic Buffer Chaining
  - Buffer Reclaim After Receipt of Bad Frames
  - Save Bad Frames, Optionally
- Interfaces to 8-bit and 16-bit Microprocessors
- 48 Pin DIP and 68 Pin PLCC

  (see "Intel Packaging" Document, Order Number: 231369-001)
- Supports Minimum Component Systems
  - Shared Bus Configuration
  - Interface to iAPX 186 and 188 Microprocessors without Glue
- Supports High Performance Systems
  - Bus Master, with On-Chip DMA
  - 5 MBytes/Sec Bus Bandwidth
  - Compatible with Dual Port Memory
  - Back to Back Frame Reception at 10 Mbps
- Network Management
  - CRC Error Tally
  - Alignment Error Tally
  - Location of Cable Faults
- Self-Test Diagnostics
  - Internal Loopback
  - External Loopback
  - Internal Register Dump
  - Backoff Timer Check

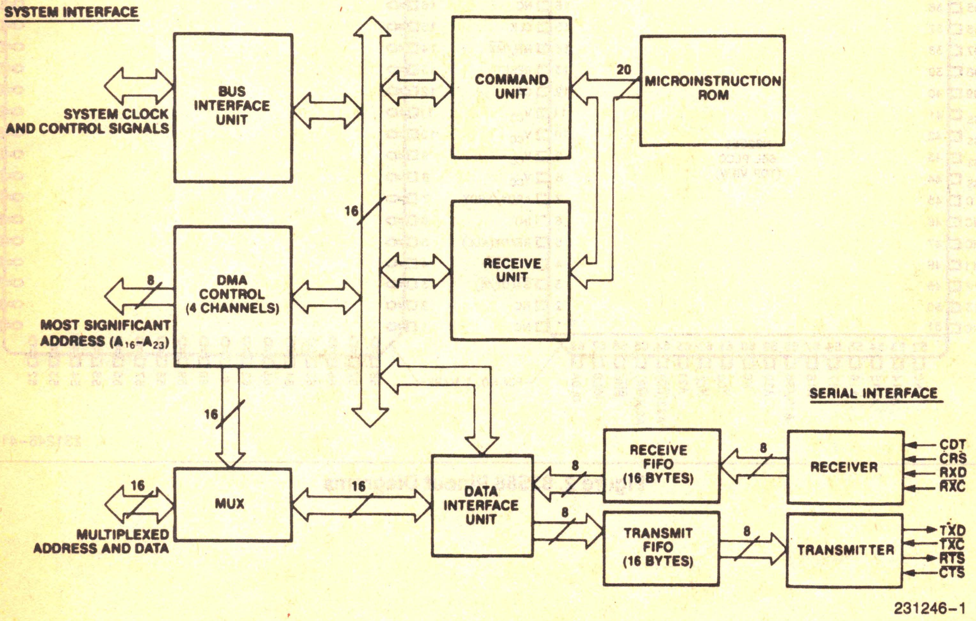
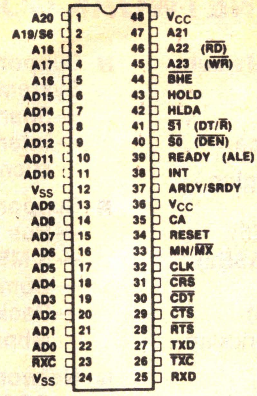


Figure 1. 82586 Functional Block Diagram

*IBM is a trademark of International Business Machines Corp.

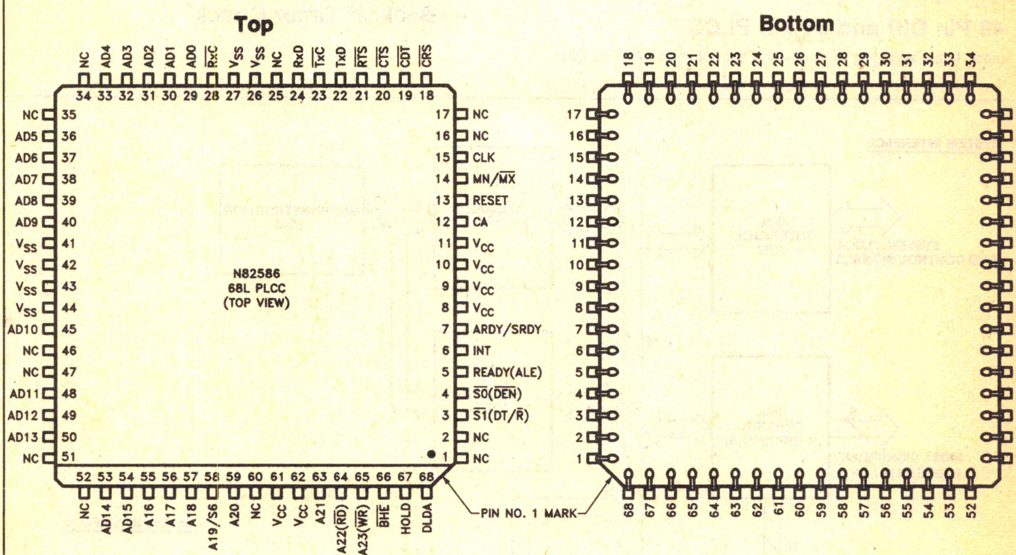
December 1986
Order Number: 231246-004



231246–2

NOTE:
The symbols in parentheses correspond to minimum mode.

Plastic Leaded Chip Carrier



231246–41

Figure 2. 82586 Pinout Diagrams

The 82586 is an intelligent, high performance Local Area Network coprocessor, implementing the CSMA/CD access method (Carrier Sense Multiple Access with Collision Detection). It performs all time-crystal functions independently of the host processor, which maximizes performance and network efficiency.

The 82586 performs the full set of IEEE 802.3 CSMA/CD media access control and channel interface functions including: framing, preamble generation and stripping, source address generation, destination address checking, CRC generation and checking, short frame detection. Any data rate up to 10 Mb/s can be used.

The 82586 features a powerful host system interface. It automatically manages memory structures with command chaining and bidirectional data chaining. An on-chip DMA controller manages 4 channels transparently to the user. Buffers containing errored or collided frames can be automatically recovered. The 82586 can be configured for 8-bit or 16-bit data path, with maximum burst transfer rate of 2 or 4 Mbyte/sec. respectively. Memory address space is 16 Mbyte maximum.

The 82586 provides two independent 16 byte FIFOs, one for receiving and one for transmitting. The threshold for block transfer to/from memory is programmable, enabling the user to optimize bus overhead for a given worst case bus latency.

The 82586 provides a rich set of diagnostic and network management functions including: internal and external loopbacks, exception condition tallies, channel activity indicators, optional capture of all frames regardless of destination address, optional capture of errored or collided frames, and time domain reflectometry for locating faults in the cable.

The 82586 can be used in either baseband or broadband networks. It can be configured for maximum network efficiency (minimum contention overhead) for any length network operating at any data rate up to 10 Mbls. The controller supports address field lengths of 1, 2, 3, 4, 5, or 6 bytes. It can be configured for either the IEEE 802.3/Ethernet or HDLC method of frame delineation. Both 16-bit and 32-bit CRC are supported.

The 82586 is fabricated in Intel's reliable HMOS II 5V technology and is available in a 48 pin DIP or 68 pin PLCC package.

Table 1. 82586 Pin Description

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type	Name and Function
V_{CC}, V_{CC}	48, 36	8, 9, 10, 11, 61, 62		System Power: +5V Power Supply.
V_{SS}, V_{SS}	12, 24	26, 27, 41, 42, 43, 44		System Ground.
RESET	34	13	I	RESET is an active HIGH internally synchronized signal, causing the 82586 to terminate present activity immediately. The signal must be HIGH for at least four clock cycles. The 82586 will execute RESET within ten system clock cycles starting from RESET HIGH. When RESET returns LOW, the 82586 waits for the first CA to begin the initialization sequence.
TxD	27	22	0	Transmitted Serial Data output signal. This signal is HIGH when not transmitting.
$\overline{TxC}$	26	23	I	Transmit Data Clock. This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ mode of operation, data is transferred to the TxD pin on the HIGH to LOW clock transition.
RxD	25	24	I	Received Data Input Signal.
$\overline{RxC}$	23	28	I	Received Data Clock. This signal provides timing information to the internal shifting logic depending upon the mode of data transfer. For NRZ data, the state of the RxD pin is sampled on the HIGH to LOW clock transition.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type	Name and Function
$\overline{RTS}$	28	21	0	Request To Send signal. When LOW, notifies an external interface that the 82586 has data to transmit. It is forced HIGH after a Reset and while the Transmit Serial Unit is not sending data.
$\overline{CTS}$	29	20	I	Active LOW Clear To Send input enables the 82586 transmitter to actually send data. It is normally used as an interface handshake to $\overline{RTS}$. This signal going inactive stops transmission. It is internally synchronized. If $\overline{CTS}$ goes inactive, meeting the setup time to $\overline{TxC}$ negative edge, transmission is stopped and $\overline{RTS}$ goes inactive within, at most, two TxC cycles.
$\overline{CRS}$	31	18	I	Active LOW Carrier Sense input used to notify the 82586 that there is traffic on the serial link. It is used only if the 82586 is configured for external Carrier Sense. When so configured, external circuitry is required for detecting serial link traffic. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles.
$\overline{CDT}$	30	19	I	Active LOW Collision Detect input is used to notify the 82586 that a collision has occurred. It is used only if the 82586 is configured for external Collision Detect. External circuitry is required for detecting the collision. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles. During transmission, the 82586 is able to recognize a collision one bit time after preamble transmission has begun.
INT	38	6	0	Active HIGH Interrupt request signal.
CLK	32	15	I	The system clock input from the 80186 or another symmetric clock generator.
MN/$\overline{MX}$	33	14	I	When HIGH, MN/$\overline{MX}$ selects $\overline{RD}$, $\overline{WR}$, ALE $\overline{DEN}$, DT/$\overline{R}$ (Minimum Mode). When LOW, MN/$\overline{MX}$ selects A22, A23, READY, $\overline{S0}$, $\overline{S1}$ (Maximum Mode). Note: This pin should be static during 82586 operation.
AD0–AD15	6–11, 13–22	29–33, 36–40, 45, 48, 49, 50, 53, 54	I/O	These lines form the time multiplexed memory address (t1) and data (t2, t3, tW, t4) bus. When operating with an 8-bit bus, the high byte will output the address during the entire cycle. AD0–AD15 are floated after a RESET or when the bus is not acquired.
A16–A18 A20–A23	1, 3–5 45–47	55–57, 59, 63–65	0	These lines constitute 7 out of 8 most significant address bits for memory operation. They switch during t1 and stay valid during the entire memory cycle. The lines are floated after RESET or when the bus is not acquired. Address lines A22 and A23 are not available for use in minimum mode.
A19/S6	2	58	0	During t1 it forms line 19 of the memory address. During t2 through t4 it is used as a status indicating that this is a Master peripheral cycle, and is HIGH. Its timing is identical to that of AD0–AD15 during write operation.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type	Name and Function
HOLD	43	67	O	HOLD is an active HIGH signal used by the 82586 to request local bus mastership at the end of the current CPU bus transfer cycle, or at the end of the current DMA burst transfer cycle. In normal operation, HOLD goes inactive before HLDA. The 82586 can be forced off the bus by HLDA going inactive. In this case, HOLD goes inactive within four clock cycles in word mode and eight clock cycles in byte mode.
HLDA	42	68	I	HLDA is an active HIGH Hold Acknowledge signal indicating that the CPU has received the HOLD request and that bus control has been relinquished to the 82586. It is internally synchronized. After HOLD is detected as LOW, the processor drives HLDA LOW. Note, CONNECTING V_{CC} TO HLDA IS NOT ALLOWED because it will cause a deadlock. Users wanting to give permanent bus access to the 82586 should connect HLDA with HOLD.
CA	35	12	I	The CA pin is a Channel Attention input used by the CPU to initiate the 82586 execution of memory resident Command Blocks. The CA signal is synchronized internally. The signal must be HIGH for at least one system clock period. It is latched internally on HIGH to LOW edge and then detected by the 82586.
$\overline{BHE}$	44	66	O	The Bus High Enable signal ($\overline{BHE}$) is used to enable data onto the most significant half of the data bus. Its timing is identical to that of A16–A23. With a 16-bit bus it is LOW and with an 8-bit bus it is HIGH. Note: after RESET, the 82586 is configured to 8-bit bus.
READY	39	5	I	This active HIGH signal is the acknowledgement from the addressed memory that the transfer cycle can be completed. While LOW, it causes wait states to be inserted. This signal must be externally synchronized with the system clock. The Ready signal internal to the 82586 is a logical OR between READY and SRDY/ARDY.
ARDY/SRDY	37	7	I	This active HIGH signal performs the same function as READY. If it is programmed at configure time to SRDY, it is identical to READY. If it is programmed to ARDY, the positive edge of the Ready signal is internally synchronized. Note, the negative edge must still meet setup and hold time specifications, when in ARDY mode. The ARDY signal must be active for at least one system clock HIGH period for proper strobing. The Ready signal internal to the 82586 is a logical OR between READY (in Maximum Mode only) and SRDY/ARDY. Note that following RESET, this pin assumes ARDY mode.

Table 1. 82586 Pin Description (Continued)

Symbol	48 Pin DIP Pin No.	68 Pin PLCC Pin No.	Type	Name and Function
$\overline{\text{SO}}$, $\overline{\text{S1}}$	40,41	4, 3	0	Maximum mode only. These status pins define the type of DMA transfer during the current memory cycle. They are encoded as follows: $\overline{\text{S1}}$ $\overline{\text{SO}}$ 0 0 Not Used 0 1 Read Memory 1 0 Write Memory 1 1 Passive Status is active from the middle of t4 to the end of t2. They return to the passive state during t3 or during tW when READY or ARDY is HIGH. These signals can be used by the 8288 Bus Controller to generate all memory control and timing signals. Any change from the passive state signals the 8288 to start the next t1 to t4 bus cycle. These pins are pulled HIGH and floated after a system RESET and when the bus is not acquired.
$\overline{\text{RD}}$	46	64	0	Used in minimum mode only. The read strobe indicates that the 82586 is performing a memory read cycle. $\overline{\text{RD}}$ is active LOW during t2, t3 and tW of any read cycle. This signal is pulled HIGH and floated after a RESET and when the bus is not acquired.
$\overline{\text{WR}}$	45	65	0	Used in minimum mode only. The write strobe indicates that the 82586 is performing a write memory cycle. $\overline{\text{WR}}$ is active LOW during t2, t3 and tW of any write cycle. It is pulled HIGH and floats after RESET and when the bus is not acquired.
ALE	39	5	0	Used in minimum mode only. Address Latch Enable is provided by the 82586 to latch the address into the 8282/8283 address latch. It is a HIGH pulse, during t1 ('clock low') of any bus cycle. Note that ALE is never floated.
$\overline{\text{DEN}}$	40	4	0	Used in minimum mode only. Data ENable is provided as output enable for the 8286/8287 transceivers in a stand-alone (no 8288) system. $\overline{\text{DEN}}$ is active LOW during each memory access. For a read cycle, it is active from the middle of t2 until the beginning of t4. For a write cycle, it is active from the beginning of t2 until the middle of t4. It is pulled HIGH and floats after a system RESET or when the bus is not acquired.
DT/$\overline{\text{R}}$	41	3	0	Used in minimum mode only. DT/$\overline{\text{R}}$ is used in non-8288 systems using an 8286/8287 data bus transceiver. It controls the direction of data flow through the Transceiver. Logically, DT/$\overline{\text{R}}$ is equivalent to $\overline{\text{S1}}$. It becomes valid in the t4 preceding a bus cycle and remains valid until the final t4 of the cycle. This signal is pulled HIGH and floated after a RESET or when the bus is not acquired.

82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's on-chip DMA capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 is optimized to interface the iAPX 186, but due to the small number of hardware signals between the 82586 and the CPU, the 82586 can operate easily with other processors. The 82586/host interaction is explained separately in terms of the logical interface and the hardware bus interface.

The 82586 consists of two independent units: Command Unit (CU) and Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two types of activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The only hardware signals that connect the CPU and the 82586 are INTERRUPT and CHANNEL ATTENTION (see Figure 3). Interrupt is used by the 82586 to draw the CPU's attention to a change in the contents of the SCB. Channel Attention is used by the CPU to draw the 82586's attention.

82586 SYSTEM MEMORY STRUCTURE

The Shared Memory structure consists of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA) (see Figure 4).

The Initialization Root is at a predetermined location in the memory space, (OFFFFF6H), known to both the host CPU and the 82586. The root is accessed at initialization and points to the System Control Block.

The System Control Block (SCB) functions as a bidirectional mail drop between the host CPU, CU and RU. It is the central element through which the CPU and the 82586 exchange control and status information. The SCB consists of two parts, the first of which entails instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands for the CU, a pointer to the receive frame area, and a set of Interrupt acknowledge bits. The second entails status information keyed by the 82586 to the CPU, including: state of the CU and RU (e.g. IDLE, ACTIVE READY, SUSPENDED, NO RECEIVE RESOURCES), interrupts bits (command completed, frame received, CU not ready, RU not ready), and statistics (see Figure 4).

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CB's contain command specific parameters and command specific statuses. Specifically, these high level commands are called Action Commands (e.g. Transmit, Configure).

A specific command, Transmit, causes transmission of a frame by the 82586. The Transmit command block includes Destination Address, Length Field, and a pointer to a list of linked buffers that holds the frame to be constructed from several buffers scattered in memory. The Command Unit performs with-

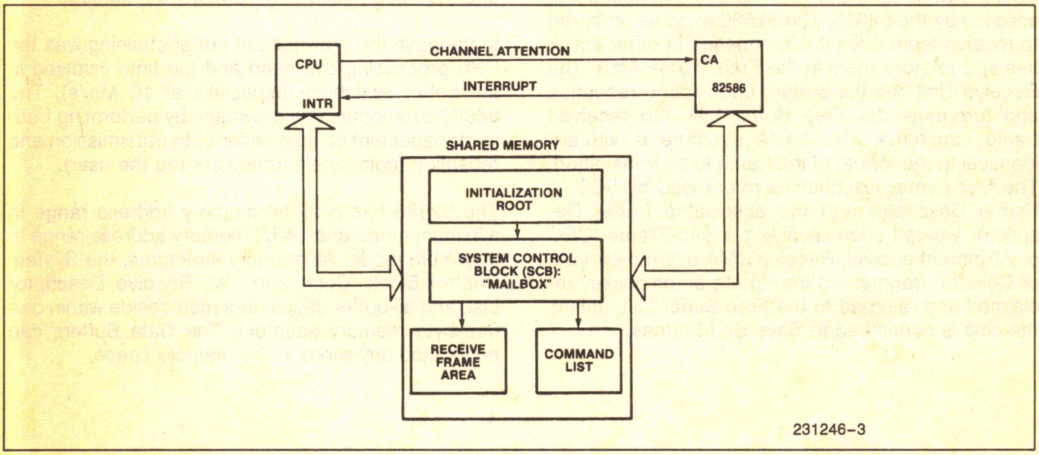


Figure 3. 82586/Host CPU Interaction

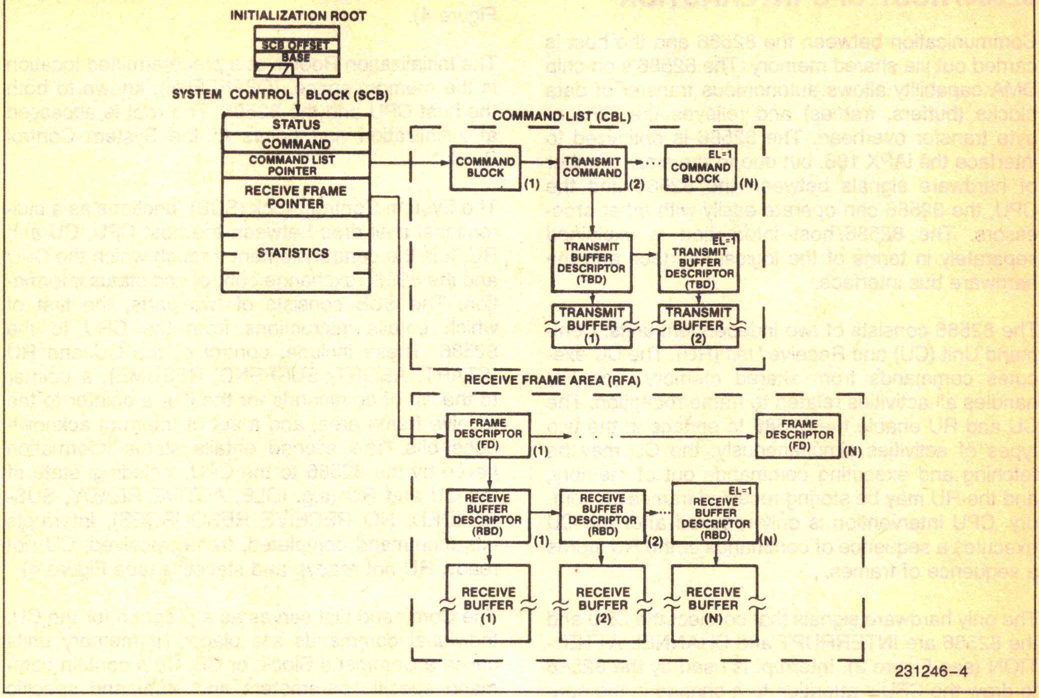


Figure 4. 82586 Shared Memory Structure

out the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers in parallel. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and a list of buffers prepared by the user. It is conceptually distinct from the Command List. Frames arrive without being solicited by the 82586. The 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB. A Frame Descriptor and the associated Buffer Descriptor wasted upon receiving a Bad Frame (CRC or Alignment errored, Receive DMA overrun errored, or Collision fragmented frame) are automatically reclaimed and returned to the Free Buffer List, unless the chip is configured to Save Bad Frames.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in Ethernet) for each frame. Taking into account that a typical frame size may be about 100 bytes, this practice is very inefficient. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was the CPU processing overhead and the time involved in the buffer switching (especially at 10 Mb/s). The 82586 overcomes this drawback by performing buffer management on its own for both transmission and reception (completely transparent to the user).

The 82586 has a 22-bit memory address range in minimum mode and 24-bit memory address range in maximum mode. All memory structures, the System Control Block, Command List, Receive Descriptor List, and all buffer descriptors must reside within one 64K-byte memory segment. The Data Buffers can be located anywhere in the memory space.

TRANSMITTING FRAMES

The 82586 executes high level action commands from the Command List in external memory. Action commands are fetched and executed in parallel with the host CPU's operation, thereby significantly improving system performance. The general action commands format is shown in Figure 5.

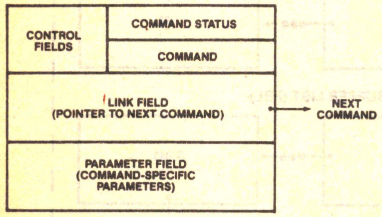


Figure 5. Action Command Format

Message transmission is accomplished by using the Transmit command. A single Transmit command contains, as part of the command-specific parameters, the destination address and length field for the transmitted frame along with a pointer to a buffer area in memory containing the data portion of the frame. (See Figure 15.) The data field is contained in a memory data structure consisting of a Buffer Descriptor (BD) and Data Buffer (or a linked list of buffer descriptors and buffers) as shown in Figure 6. The BD contains a Link Field which points to the next BD on the list and a 24-bit address pointing to the Data Buffer itself. The length of the Data Buffer is specified by the Actual Count field of the BD.

Using the BD's and Data Buffers, multiple Data Buffers can be 'chained' together. Thus, a frame with a long Data Field can be transmitted using multiple (shorter) Data buffers chained together. This chaining technique allows the system designer to develop efficient buffer management policies.

The 82586 automatically generates the preamble (alternating 1's and 0's) and start frame delimiter, fetches the destination address and length field from the Transmit command, inserts its unique address as the source address, fetches the data field from

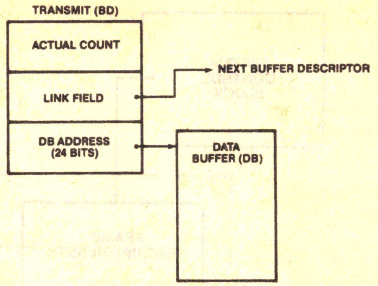


Figure 6. Data Buffer Descriptor and Data Buffer Structure

buffers pointed to by the Transmit command, and computes and appends the CRC at the end of the frame. See Figure 7.

The 82586 can be configured to generate either the Ethernet or HDLC start and end frame delimiters. In the Ethernet mode, the start frame delimiter is 10101011 and the end frame delimiter indicated by the lack of a signal after transmitting the last bit of the frame check sequence field. When in the HDLC mode, the 82586 will generate the 01111110 'flag' for the start and end frame delimiters and perform the standard 'bit stuffing/stripping'. In addition, the 82586 will optionally pad frames that are shorter than the specified minimum frame length by appending the appropriate number of flags to the end of the frame.

In the event of a collision (or collisions), the 82586 manages the entire jam, random wait and retry process, reinitializing DMA pointers without CPU intervention. Multiple frames can be sent by linking the appropriate number of Transmit commands together. This is particularly useful when transmitting a message that is larger than the maximum frame size (1518 bytes for Ethernet).

RECEIVING FRAMES

In order to minimize CPU overhead, the 82586 is designed to receive frames without CPU supervision. The host CPU first sets aside an adequate

PREAMBLE	START FRAME DELIMITER	DEST ADDR	SOURCE ADDR	LENGTH FIELD	DATA FIELD	FRAME CHECK SEQUENCE	END FRAME DELIMITER

Figure 7. Frame Format

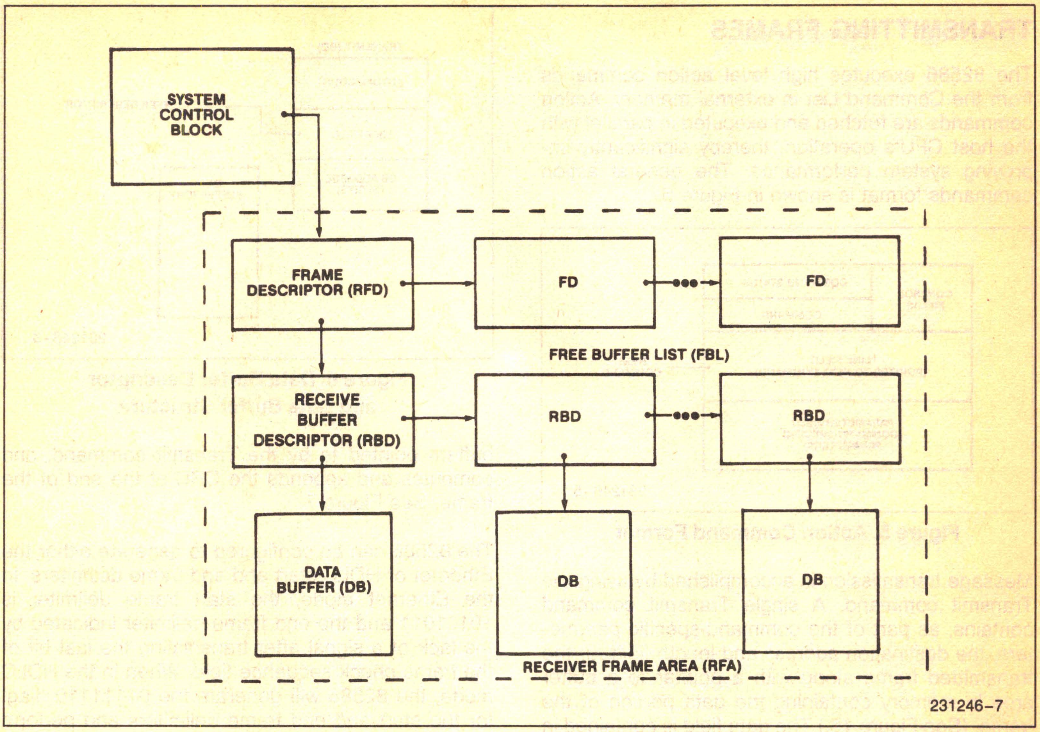


Figure 8. Receive Frame Area Diagram

amount of receive buffer space and then enables the 82586's Receive Unit. Once enabled, the RU 'watches' for any of its frames which it automatically stores in the Receive Frame Area (RFA). The RFA consists of a Receive Descriptor List (RDL) and a list of free buffers called the Free Buffer List (FBL) as shown in Figure 8. The individual Receive Frame Descriptors that make up the RDL are used by the 82586 to store the destination and source address, length field and status of each frame that is received. (Figure 9.)

The 82586, once enabled, checks each passing frame for an address match. The 82586 will recognize its own unique address, one or more multicast addresses or the broadcast address. If a match occurs, it stores the destination and source address and length field in the next available RFD. It then begins filling the next free Data Buffer on the FBL (which is pointed to by the current RFD) with the data portion of the incoming frame. As one DB is filled, the 82586 automatically fetches the next DB on the FBL until the entire frame is received. This buffer chaining technique is particularly memory efficient because it allows the system designer to set aside buffers that fit a frame size that may be much shorter than the maximum allowable frame.

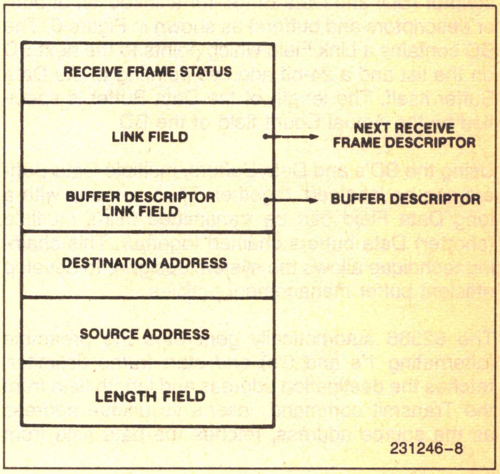


Figure 9. Receive Frame Descriptor

Once the entire frame is received without error, the 82586 performs the following housekeeping tasks:

- Updates the Actual Count field of the last Buffer Descriptor used to hold the frame just received with the number of bytes stored in its associated Data Buffer.

- Fetches the address of the next free Receive Frame Descriptor.
- Writes the address of the next free Buffer Descriptor into the next free Receive Frame Descriptor.
- Posts a 'Frame Received' interrupt status bit in the SCB.
- Interrupts the CPU.

In the event of a frame error, such as a CRC error, the 82586 automatically reinitializes its DMA pointers and reclaims any data buffers containing the bad frame. As long as Receive Frame Descriptors and data buffers are available, the 82586 will continue to receive frames without further CPU help.

82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. It is particularly difficult to pinpoint a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 reports on the following events after each frame transmitted:

- Transmission successful.
- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun because the system bus did not keep up with the transmission.
- Transmission unsuccessful; number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame.
- Alignment error: incorrect CRC in a misaligned frame.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

NETWORK PLANNING AND MAINTENANCE

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a network management entity.

Network Activity information is provided in the status of each frame transmitted. The activity indicators are:

- Number of collisions: number of collisions the 82586 experienced in attempting to transmit this frame.
- Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes the address filtering, and is longer than the Minimum Frame Length configuration parameter.

- CRC errors: number of frames that experienced a CRC error and were properly aligned.
- Alignment errors: number of frames that experienced a CRC error and were misaligned.
- No-resources: number of correct frames lost due to lack of memory resources.
- Overrun errors: number of frame sequences lost due to DMA overrun.

The 82586 can be configured to Promiscuous Mode. In this mode it captures all frames transmitted on the Network without checking the Destination Address. This is useful in implementing a monitoring station to capture all frames for analysis.

The 82586 is capable of determining if there is a short or open circuit anywhere in the Network using the built in Time Domain Reflectometer (TDR) mechanism.

STATION DIAGNOSTICS

The chip can be configured to External Loopback. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Serial Interface Unit, the Transceiver cable, or in the Transceiver.

The 82586 has a mechanism recognizing the transceiver 'heart beat' signal for verifying the correct operation of the Transceiver's collision detection circuitry.

82586 SELF TESTING

The 82586 can be configured to Internal Loopback. It disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock.

The Dump Command causes the chip to write over 100 bytes of its internal registers to memory.

The Diagnose command checks the exponential Backoff random number generator internal to the 82586.

CONTROLLING THE 82586

The CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU) of the 82586 via the System Control Block.

THE COMMAND UNIT (CU)

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE—CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- SUSPENDED—CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- ACTIVE—CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command.

THE RECEIVE UNIT (RU)

The Receive Unit is the logical unit that receives frames and stores them in memory.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE—RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- NO-RESOURCES—RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- SUSPENDED—RU has free memory resources to store incoming frames but discard them anyway.

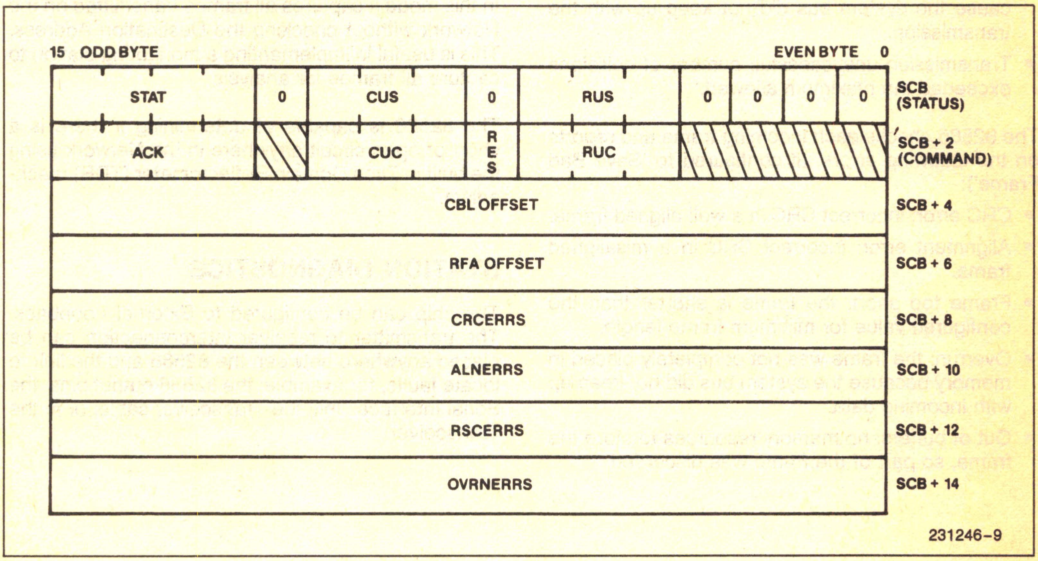


Figure 10. System Control Block (SCB) Format

- READY—RU has free memory resources and stores incoming frames.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in Frame Descriptor, FD, COMMAND word of the frame currently being received, or setting EL bit of Buffer Descriptor, BD, of the buffer currently being filled.

SYSTEM CONTROL BLOCK (SCB)

The System Control Block is the communication mail-box between the 82586 and the host CPU. The SCB format is shown in Figure 10.

The host CPU issues Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. After the required Control Command is setup, the CPU sends a CA signal to the 82586.

SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

STATUS word: Indicates the status of the 82586. This word is modified only by the 82586. Defined bits are:

CX	(Bit 15)	• A command in the CBL having its 'I' (interrupt) bit set has been executed.
FR	(Bit 14)	• A frame has been received.
CNR	(Bit 13)	• The Command Unit left the Active state.
RNR	(Bit 12)	• The Receive Unit left the Ready state.
CUS	(Bits 8–10)	• (3 bits) this field contains the status of the Command Unit. Valid values are: 0 — Idle 1 — Suspended 2 — Active 3-7 — Not Used
RUS	(Bits 4—6)	• (3 bits) this field contains the status of the Receive Unit. Valid values are: 0 — Idle 1 — Suspended 2 — No Resources 3 — Not Used 4 — Ready 5-7 — Not Used

COMMAND word: Specifies the action to be performed as a result of the CA. This word is set by the CPU and cleared by the 82586. Defined bits are:

ACK-CX	(Bit 15)	• Acknowledges the command executed event.
ACK-FR	(Bit 14)	• Acknowledges the frame received event.
ACK-CNA	(Bit 13)	• Acknowledes that the Command Unit became not ready.
ACK-RNR	(Bit 12)	• Acknowledges that the Receive Unit became not ready.
CUC	(Bits 8–10)	• (3 bits) this field contains the command to the Command Unit.
	0	• NOP (doesn't affect current state of the unit).
	1	• Start execution of the first command on the CBL. If a command is in execution, then complete it before starting the new CBL. The beginning of the CBL is in CBL OFFSET.
	2	• Resume the operation of the command unit by executing the next command. This operation assumes that the command unit has been previously suspended.
	3	• Suspend execution of commands on CBL after current command is complete.
	4	• Abort execution of commands immediately.
	5–7	• Reserved, illegal for use.
RUC	(Bits 4–6)	• (3 bits) This field contains the command to the receive unit. Valid values are:
	0	• NCP (does not alter current state of unit).
	1	• Start reception of frames. If a frame is being received, then complete reception before starting. The beginning of the RFA is contained in the RFA OFFSET.
	2	• Resume frame receiving (only when in suspended state.)
	3	• Suspend frame receiving. If a frame is being received, then complete its reception before suspending.
	4	• Abort receiver operation immediately.
	5–7	• Reserved, illegal for use.
RESET	(Bit 7)	• Reset chip (logically the same as hardware RESET).

CBL-OFFSET:

gives the 16-bit offset address of the first command (Action Command) in the command list to be executed following CU-START. Thus, the 82586 reads this word only if the CUC field contained a CU-START Control Command.

RFA-OFFSET:

Points to the first Receive Frame Descriptor in the Receive Frame Area.

CRCERRS:

CRC Errors - contains the number of properly aligned frames received with a CRC error.

ALNERRS:

Alignment Errors - contains the number of misaligned frames received with a CRC error.

RSCERRS:

Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or received frame descriptors).

OVRNERRS:

Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available in time to transfer them.

ACTION COMMANDS

The 82586 executes a 'program' that is made up of action commands in the Command List. As shown in Figure 5, each command contains the command field, status and control fields, link to the next action command in the CL, and any command-specific parameters. This command format is called the Command Block.

The 82586 has a repertoire of 8 commands:

NOP
Setup Individual Address
Configure
Setup Multicast Address
Transmit
TDR
Diagnose
Dump

NOP

This command results in no action by the 82586, except as performed in normal command processing. It is present to aid in Command List manipulation.

NOP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• NOP = 0

LINK OFFSET: Address of next Command Block

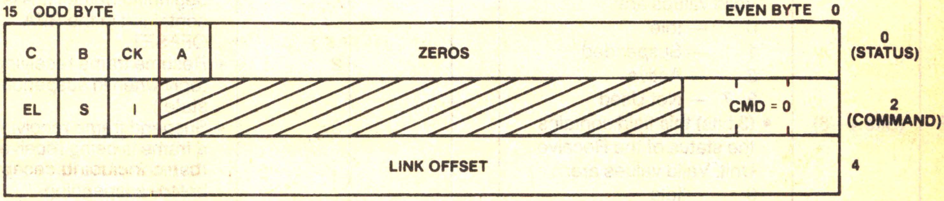


Figure 11. The NOP Command Block

IA-SETUP

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

The IA-SETUP command includes the following fields:

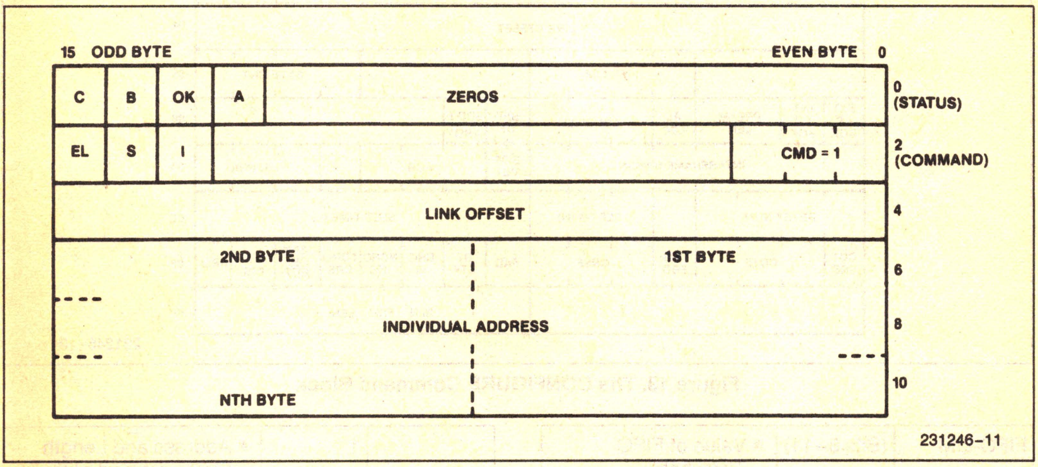


Figure 12. The IA-SETUP Command Block

STATUS word (written by 82586).

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• IA-SETUP = 1

LINK OFFSET: Address of next Command Block

INDIVIDUAL ADDRESS: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3/Ethernet. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

CONFIGURE

The CONFIGURE command is used to update the 82586 operating parameters.

The CONFIGURE command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• Configure = 2

LINK OFFSET: Address of next Command Block

Byte 6–7:

BYTE CNT	(Bits 0–3)	• Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12.

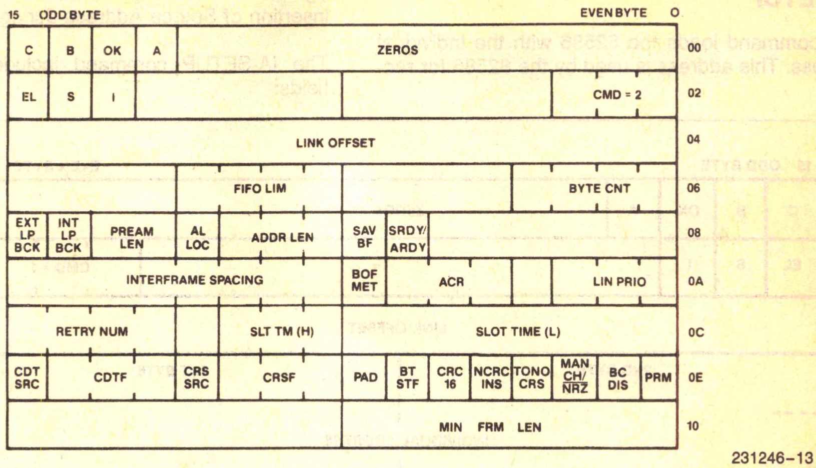


Figure 13. The CONFIGURE Command Block

FIFO-LIM	(Bits 8–11)	• Value of FIFO Threshold.

Byte 8–9:

SRDY/ARDY	(Bit 6) 0	• SRDY/ARDY pin operates as ARDY (internal synchronization).
	1	• SRDY/ARDY pin operates as SRDY (external synchronization).
SAV-BF	(Bit 7) 0	• Received bad frames are not saved in memory.
	1	• Received bad frames are saved in memory.
ADD-LEN	(Bits 8–10)	• Number of address byes. NOTE: 7 is interpreted as 0.
AL-LOC	(Bit 11) 0	• Address and Length Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted Frame, Source Address is inserted by the 82586.
	1	• Address and Length Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).
PREAM-LEN	(Bits 12–13)	• Preamble Length including Beginning of Frame indicator: 00 - 2 bytes 01 - 4 bytes 10 - 8 bytes 11 - 16 bytes
INT-LPBCK	(Bit 14)	• Internal Loopback
EXT-LPBCK	(Bit 15)	• External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.

Byte 10–11:

LIN-PRIO	(Bits 0–2)	• Linear Priority
ACR	(Bits 4–6)	• Accelerated Contention Resolution (Exponential Priority)
BOF-MET	(Bit 7)	• Exponential Backoff Method 0 - IEEE 802.3/Ethernet 1 - Alternate Method

INTER FRAME SPACING	(Bits 8–15)	• Number indicating the Interframe Spacing in TxC period units.

Byte 12–13:

SLOT-TIME (L)	(Bits 0–7)	• Slot Time Number, Low Byte
SLT-TM (H)	(Bits 8–10)	• Slot Time Number, High Bits
RETRY-NUM	(Bits 12–15)	• Maximum Number of Transmission Retries on Collisions

Byte 14–15:

PRM	(Bit 0)	• Promiscuous Mode
BC-DIS	(Bit 1)	• Broadcast Disable
MANCH/$\overline{\text{NRZ}}$	(Bit 2)	• Manchester or NRZ
		• Encoding/Decoding
	0	• NRZ
	1	• Manchester
TONO-CRS	(Bit 3)	• Transmit on No Carrier Sense
	0	• Cease Transmission if $\overline{\text{CRS}}$ Goes Inactive During Frame Transmission
	1	• Continue Transmission Even if no Carrier Sense
NCRC-INS	(Bit 4)	• No CRC Insertion
CRC-16	(Bit 5)	• CRC Type:
	0	• 32 bit Autodin II CRC Polynomial
	1	• 16 bit CCITT CRC Polynomial
BT-STF	(Bit 6)	• Bitstuffing:
	0	• End of Carrier Mode (Ethernet)
	1	• HDLC like Bitstuffing Mode
PAD	(Bit 7)	• Padding
	0	• No Padding
	1	• Perform Padding by Transmitting Flags for Remainder of Slot Time
CRSF	(Bits 8–9)	• Carrier Sense Filter in Bit Times
CRS-SRC	(Bit 11)	• Carrier Sense Source
	0	• External
	1	• Internal

CDTF	(Bits 12–14	• Collision Detect Filter in Bit Times
CDT-SRC	(Bit 15)	• Collision Detect Source
	0	• External
	1	• Internal

Byte 16:

MIN-FRM-	(Bits 0–7)	• Minimum Number of Bytes in a Frame

CONFIGURATION DEFAULTS

The default values of the configuration parameters are compatible with the IEEE 802.3/Ethernet Standards. RESET configures the 82586 according to the defaults shown in Table 2.

Table 2. 82586 Default Values

Preamble Length	=	2
Address Length	=	6
Broadcast Disable	=	0
CRC-16/CRC-32	=	0
No CRC Insertion	=	0
Bitstuffing/EOC	=	0
Padding	=	0
Min-Frame-Length	=	64
Interframe Spacing	=	96
Slot Time	=	512
Number of Retries	=	15
Linear Priority	=	0
Accelerated Contention Resolution	=	0
Exponential Backoff Method	=	0
Manchester/$\overline{\text{NRZ}}$	=	0
Internal CRS	=	0
CRS Filter	=	0
Internal CDT	=	0
CDT Filter	=	0
Transmit On No CRS	=	0
FIFO THRESHOLD	=	8
SRDY/$\overline{\text{ARDY}}$	=	0
Save Bad Frame	=	0
AT-LOC	=	0
INT Loopback	=	0
EXT Loopback	=	0
Promiscuous Mode	=	0

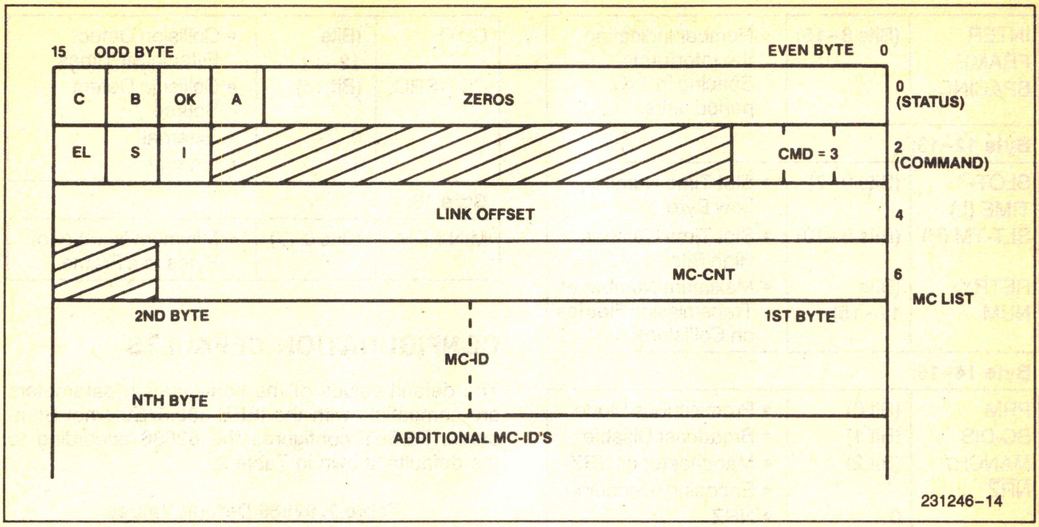


Figure 14. The MC-SETUP Command Block

MC-SETUP

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

The MC-SETUP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• MC-SETUP = 3

LINK OFFSET: Address of next Command Block

MC-CNT: A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT=0 disables reception of any incoming frame with a Multicast Address.

MC-LIST: A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2–7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

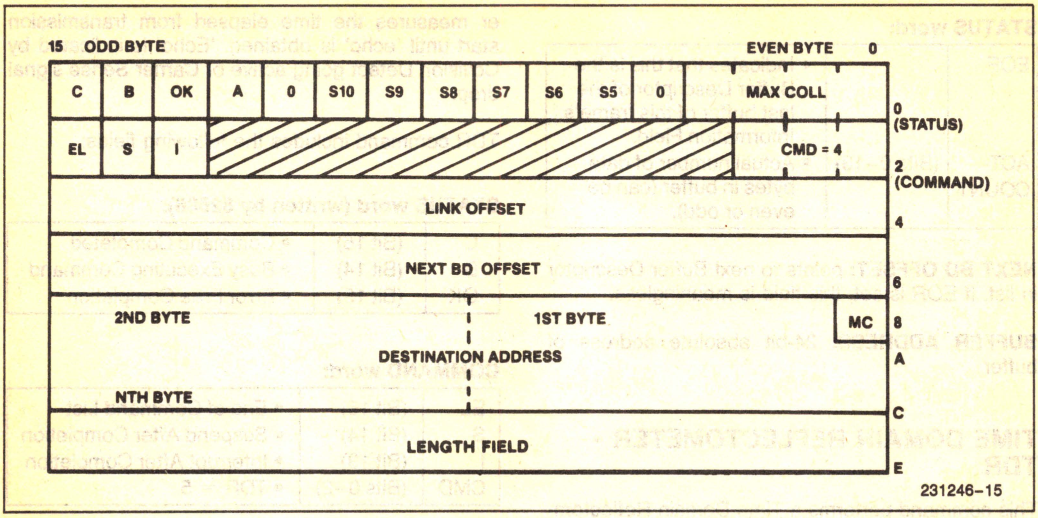


Figure 15. The Transmit Command Block

TRANSMIT

The TRANSMIT command causes transmission (and if necessary retransmission) of a frame.

TRANSMIT CB includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
A	(Bit 12)	• Command Aborted
S10	(Bit 10)	• No Carrier Sense signal during transmission (between beginning of Destination Address and end of Frame Check Sequence).
S9	(Bit 9)	• Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal.
S8	(Bit 8)	• Transmission unsuccessful (stopped) due to DMA underrun, (i.e. data not supplied from the sytem for transmission).
S7	(Bit 7)	• Transmission had to Defer to traffic on the link.
S6	(Bit 6)	• Heart Beat, indicates that during Interframe Spacing period after the previous transmission, a pulse was detected on the Collision Detect pin.
S5	(Bit 5)	• Transmission attempt stopped due to number of collisions exceeding the maximum number of retries.
MAX-COLL	(Bits 3–0)	• Number of Collisions experienced by this frame. S5 = 1 and MAX-COLL = 0 indicates that there were 16 collisions.
COMMAND word:		
EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• TRANSMIT = 4

LINK OFFSET: Address of next Command Block

TBD OFFSET: Address of list of buffers holding the information field. TBD-OFFSET = 0FFFFH indicates that there is no Information field.

DESTINATION ADDRESS: Destination Address of the frame.

LENGTH FIELD: Length field of the frame.

STATUS word:

EOF		• Indicates that this is the Buffer Descriptor of the last buffer of this frame's Information Field.
ACT-COUNT	(Bits 0–13)	• Actual number of data bytes in buffer (can be even or odd).

NEXT BD OFFSET: points to next Buffer Descriptor in list. If EOR is set, this field is meaningless.

BUFFER ADDRESS: 24-bit absolute address of buffer.

TIME DOMAIN REFLECTOMETER - TDR

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify shorts or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• TDR = 5

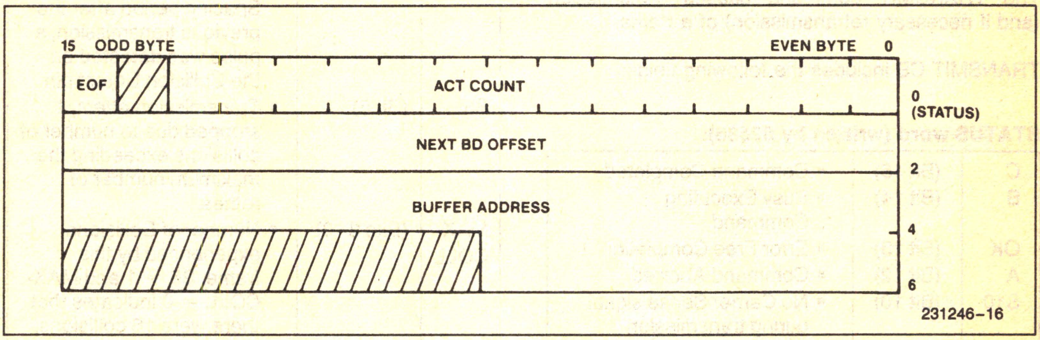


Figure 16. The Transmit Buffer Description

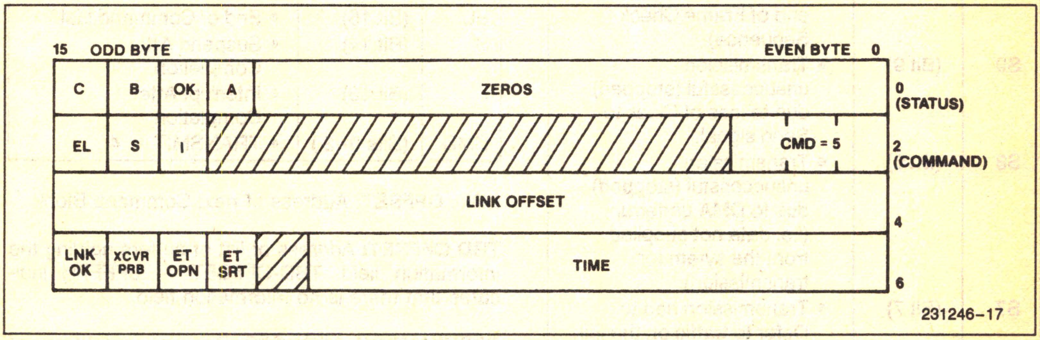


Figure 17. The TDR Command Block

LINK OFFSET: Address of next Command Block

RESULT word:

LNK-OK	(Bit 15)	• No Link Problem Identified
XCVR-PRB	(Bit 14)	• Transceiver Cable Problem identified (valid only in the case of a Transceiver that does not return Carrier Sense during transmission).
ET-OPN	(Bit 13)	• Open on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
ET-SRT	(Bit 12)	• Short on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
TIME	(Bits 0–10)	• Specifying the distance to a problem on the link (if one exists) in transmit clock cycles.

DUMP

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

DUMP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• DUMP = 6

LINK OFFSET: Address of next Command Block

BUFFER OFFSET: This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

DUMP AREA FORMAT

Figure 18 shows the format of the DUMP area. The fields are as follows:

Bytes 00H to 0AH: These bytes correspond to the 82586 CONFIGURE command field.

Bytes 0CH to 11H: The Individual Address Register content. IARO is the Individual Address least significant byte.

Bytes 12H to 13H: Status word of last command block (only bits 0–13).

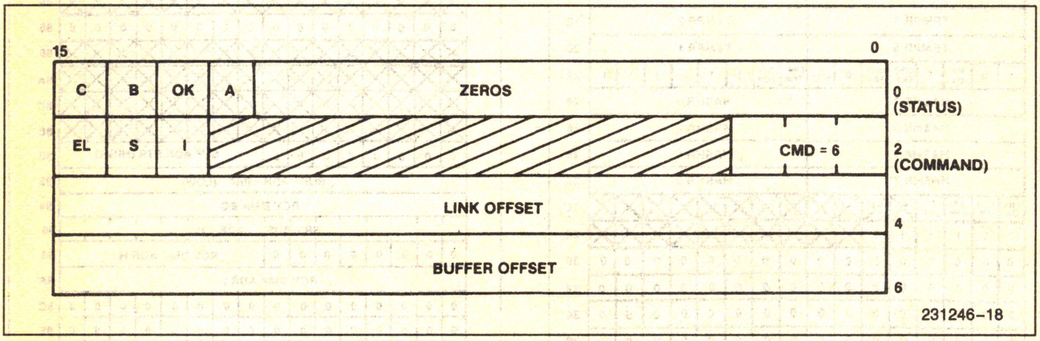


Figure 18. The DUMP Command Block

Bytes 14H to 17H: Content of the Transmit CRC generator. TXCRCRO is the least significant byte. The contents are dependent on the activity before the DUMP command:

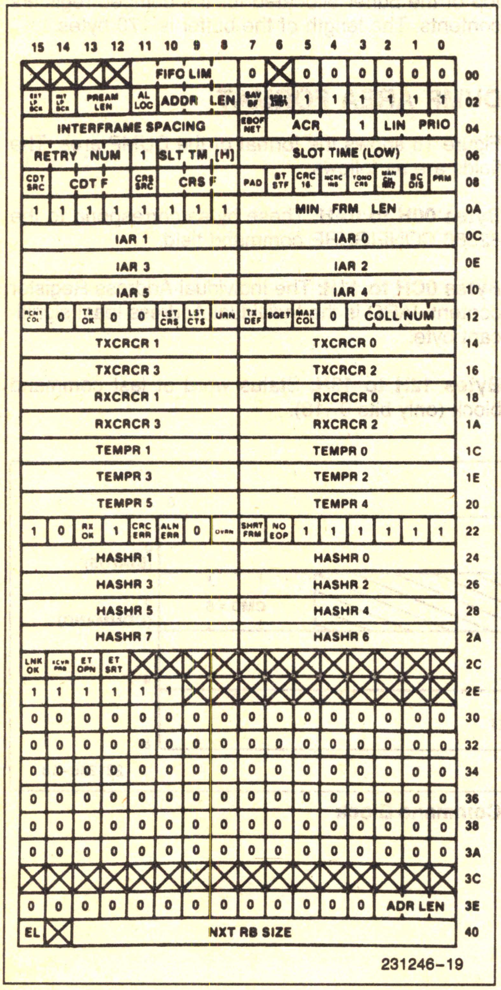
After RESET - 'All Ones.'

After successful transmission - 'All Zeros'.

After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.

After unsuccessful transmission, depends on where it stopped.

NOTE:
For 16-bit CRC only TXCRCRO and TXCRCR1 are valid.



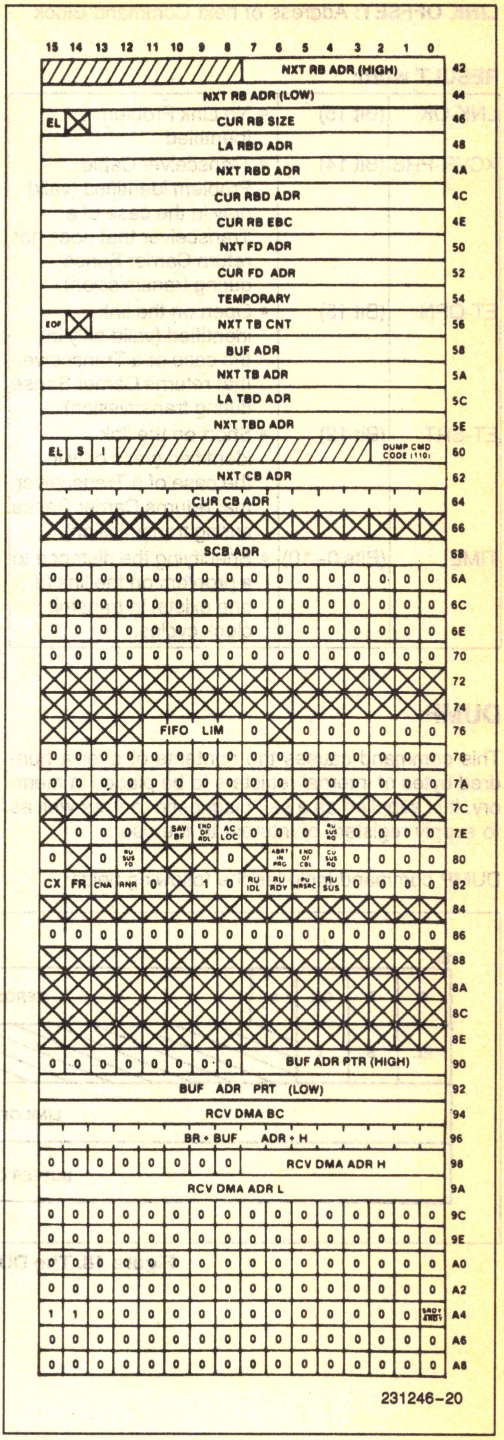


Figure 19. The DUMP Area

Bytes 18H to 1BH: Contents of Receive CRC Checker. RXCRCRO is the least significant byte. The contents are dependent on the activity performed before the DUMP command:

After RESET - 'All Ones.'

After good frame reception—

1. For CRC-CCITT - 0ID0FH
2. For CRC-Autodin-II - C704DD7BH

After Bad Frame reception - corresponds to the received information.

After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

NOTE:
Any frame on the serial link modifies this register contents.

Bytes 1CH to 21H: Temporary Registers.

Bytes 22H to 23H: Receive Status Register. Bits 6, 7, 8, 10, 11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

Bytes 24H to 2BH: HASH TABLE.

Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of bytes of available in the N + 1 buffer. EL - The ELK bit of the Receive Buffer Descriptor.

NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N + 1 Receive-Buffer Descriptor, i.e. the pointer to the N + 1 Receive Buffer.

CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame. EL - The EL bit of the last buffer in the last received frame.

LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N + 2 Receiver Buffer Descriptor.

NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N + 1 Receive Buffer Descriptor.

CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but point to Nth Receive Buffer Descriptor.

CUR-RB-EBC: Current Receive Buffer Empty Byte Count Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C = 1 and B = 0, then NXT-FD-ADR is the address of N + 2 Receive Frame Descriptor (with B = C = 0) and is equal to the LINK-ADDRESS field in N + 1 Receive Frame Descriptor.

CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N + 1 Receive Frame Descriptor (with B = 1, C = 0).

Bytes 54H to 55H: Temporary register.

NXT-TB-CNT: Next Transmit Buffer Count. Let N be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the Nth Transmit Buffer Descriptor. EOF - Corresponds to the EOF bit of the Nth Transmit Buffer Descriptor. EOF = 1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the Nth buffer address.

LA-TB-ADR: Look Ahead Transmit Buffer Descriptor Address. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the Nth Buffer Descriptor.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor N-1. Actually, it is the address of Transmit Buffer Descriptor N.

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request.

END-OF-CBL (Bit 5) - End of Command Block List. If "1" indicates that DUMP-STATUS is the last command in the command chain.

ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.

RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume N is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of N + 1 Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.

RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.

RU-RDY (Bit 6) - Receive Unit in READY state.

RU-IDL (Bit 7) - Receive Unit in IDLE state.

RNR (Bit 12) - RNR Interrupt in Service bit.

CNA (Bit 13) - CNA Interrupt in Service bit.

FR (Bit 14) - FR Interrupt in Service bit.

CX (Bit 15) - CX Interrupt in Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.
2. If AT-LOCation = 1 then it contains the size of the next Receive Buffer.

BR + BUF − PTR + 96H - Sum of Base Address plus BUF − PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.
2. If AT-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.

The following nomenclature has been used in the DUMP table:

0	• The 82586 writes zero in this location.
1	• The 82586 writes one in this location.
X	• The 82586 writes zero or one in this location.
///	• The 82586 copies this location from the corresponding position in the memory structure.

DIAGNOSE

The DIAGNOSE Command triggers an internal self test procedure of backoff related registers and counters.

The DIAGNOSE command includes the following:

STATUS word (written by 82586):

C	(Bit 15)	• Command Completed
B	(Bit 14)	• Busy Executing Command
OK	(Bit 13)	• Error Free Completion
FAIL	(Bit 11)	• Indicates that the Self Test Procedured Failed

COMMAND word:

EL	(Bit 15)	• End of Command List
S	(Bit 14)	• Suspend After Completion
I	(Bit 13)	• Interrupt After Completion
CMD	(Bits 0–2)	• DIAGNOSE = 7

LINK OFFSET: Address of next Command Block.

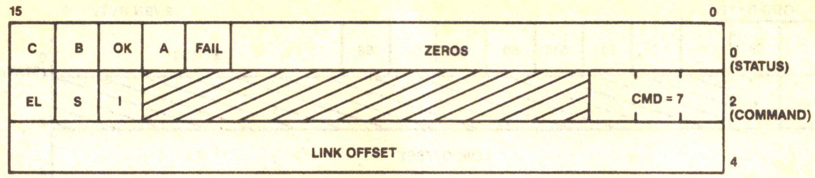


Figure 20. The DIAGNOSE Command Block

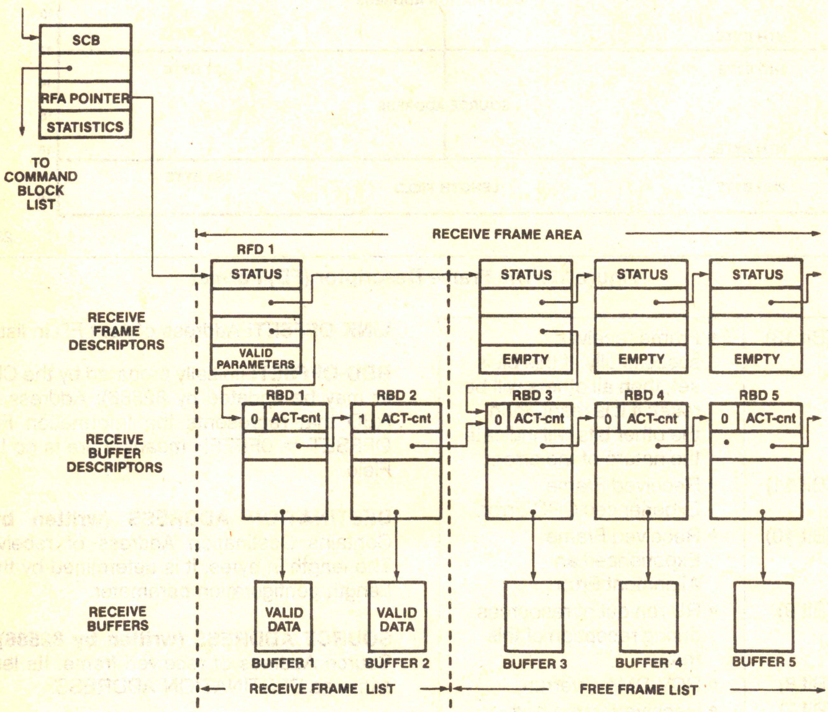


Figure 21. The Receive Frame Area

RECEIVE FRAME AREA (RFA)

The Receive Frame Area, RFA, is prepared by the host CPU, data is placed into the RFA by the 82586 as frames are received. RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. RFA-OFFSET field of SCB points to the first FD of the chain; the last FD is identified by the End-of-Listing flag (EL). See Figure 21.

FRAME DESCRIPTOR (FD) FORMAT

The FD includes the following fields:

STATUS word (set by the 82586):

C	(Bit 15)	• Completed Storing Frame.
B	(Bit 14)	• FD was Consumed by RU.

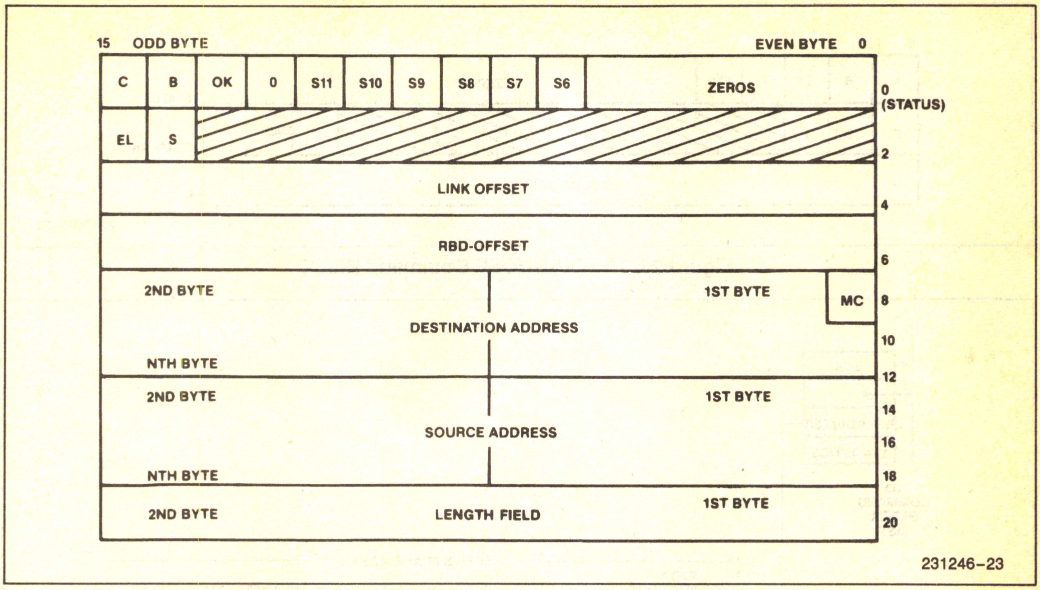


Figure 22. The Frame Descriptor (FD) Format

OK	(Bit 13)	• Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.
S11	(Bit 11)	• Received Frame Experienced CRC Error.
S10	(Bit 10)	• Received Frame Experienced an Alignment Error.
S9	(Bit 9)	• RU ran out of resources during reception of this frame.
S8	(Bit 8)	• RCV-DMA Overrun.
S7	(Bit 7)	• Received frame had fewer bits than configured Minimum Frame Length.
S6	(Bit 6)	• No EOF flag detected (only when configured to Bitstuffing).

COMMAND word:

EL	(Bit 15)	• Last FD in the List.
S	(Bit 14)	• RU should be suspended after receiving this frame.

LINK OFFSET: Address of next FD in list.

RBD-OFFSET: (initially prepared by the CPU and later may be updated by 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means there is no Information Field.

DESTINATION ADDRESS (written by 82586): Contains Destination Address of received frame. The length in bytes, it is determined by the Address Length configuration parameter.

SOURCE ADDRESS (written by 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS.

LENGTH FIELD (written by 82586): Contains the 2 byte Type Field of received frame.

RECEIVE BUFFER DESCRIPTOR FORMAT

The Receive Buffer Descriptor (RBD) holds information about a buffer; size and location, and the means for forming a chain of RBDs, (forward pointer and end-of-frame indication).

The Buffer Descriptor contains the following fields.

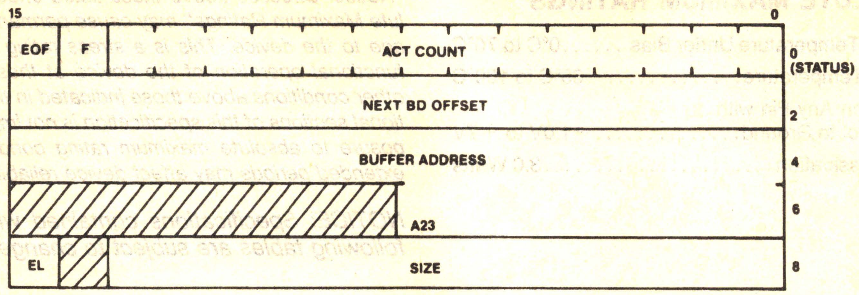


Figure 23. The Receive Buffer Descriptor (RBD) Format

STATUS word (written by the 82586).

EOF	(Bit 15)	• Last buffer in received frame.
F	(Bit 14)	• ACT COUNT field is valid.
ACT COUNT	(Bits 0–13)	• Number of bytes in the buffer that are actually occupied.

NEXT RBD OFFSET: Address of next BD in list of BD's.

BUFFER ADDRESS: 24-bit absolute address of buffer.

EL/SIZE:

EL	(BIT 15)	• Last BD in list.
SIZE	(Bits 0–13)	• Number of bytes the buffer is capable of holding.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...... 0°C to 70°C
Storage Temperature .... −65°C to 150°C
Voltage on Any Pin with
Respect to Ground .... −1.0V to +7V
Power Dissipation .... 3.0 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

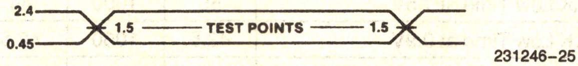
T_A = 0°C to 70°C, T_C = 0°C to 105°C, V_{CC} = 5V ±10%, CLK has MOS levels (See V_{MIL}, V_{MIH}, V_{MOL}, V_{MOH}). $\overline{TxC}$ and $\overline{RxC}$ have 82501 compatible levels (V_{MIL}, V_{TIH}, V_{RIH}). All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, $_{OH}$).

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (TTL)	−0.5	+0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL} = 2.5 mA
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH} − 400 μA
V_{MIL}	Input Low Voltage (MOS)	−0.5	0.6	V	
V_{MIH}	Input High Voltage (MOS)	3.9	V_{CC} + 0.5	V	
V_{TIH}	Input High Voltage ($\overline{TxC}$)	3.3	V_{CC} + 0.5	V	
V_{RIH}	Input High Voltage ($\overline{RxC}$)	3.0	V_{CC} + 0.5	V	
V_{MOL}	Output Low Voltage (MOS)		0.45	V	I_{OL} 2.5 mA
V_{MOH}	Output High Voltage (MOS)	V_{CC} − 0.5		V	I_{OH} − 400 μA
I_{LI}	Input Leakage Current		±10	μA	0 ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45 ≤ V_{OUT} ≤ V_{CC}
C_{IN}	Capacitance of Input Buffer		10	pF	FC = 1 MHz
C_{OUT}	Capacitance of Output Buffer		20	pF	FC = 1 MHz
I_{CC}	Power Supply Current		550 450	mA	T_A = 0°C T_A = 70°C

SYSTEM INTERFACE A.C. TIMING CHARACTERISTICS

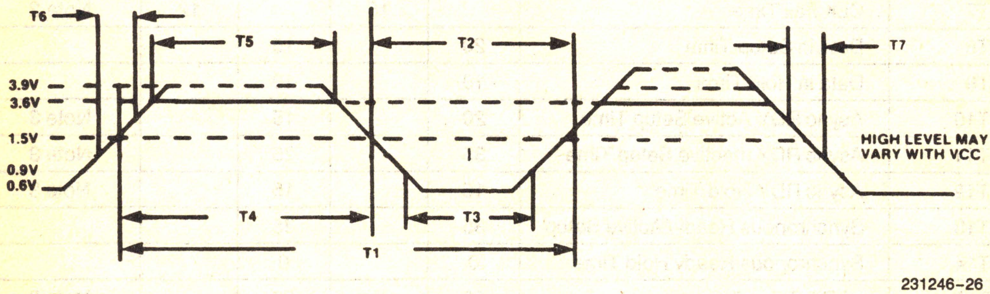
T_A = 0°C to 70°C, T_C = 0°C to 105°C, V_{CC} = 5V ±10%. Figures 24 and 25 define how the measurements should be done.

INPUT AND OUTPUT WAVEFORMS FOR A.C. TESTS



AC Testing Inputs are Driven at 2.4V for a Logic 1 and 0.45 for a Logic 0. Timing measurements are made at 1.5V for both a Logic 1 and 0

Figure 24. TTL Input/Output Voltage Levels for Timing Measurements



MOS I/O measurements are taken at 0.1 and 0.9 of the voltage swing

Figure 25. System Clock CMOS Input Voltage Levels for Timing Measurements

INPUT TIMING REQUIREMENTS*

Symbol	Parameter	82586 (8 MHz)		82586-10 (10 MHz)		Comments
		Min	Max	Min	Max	
T1	CLK Cycle Period	125	2000	100	200	
T2	CLK Low Time at 1.5V	55	1000	44	1000	
T3	CLK Low Time at 0.9V	42.5	1000	42.5	1000	
T4	CLK High Time at 1.5V	55		44		
T5	CLK High Time at 3.6V	42.5		42.5		
T6	CLK Rise Time		15		12	Note 1
T7	CLK Fall Time		15		12	Note 2
T8	Data in Setup Time	20		15		
T9	Data in Hold Time	10		10		
T10	Async RDY Active Setup Time	20		15		Note 3
T11	Async RDY Inactive Setup Time	35		25		Note 3
T12	Async RDY Hold Time	15		15		Note 3
T13	Synchronous Ready/Active Setup	35		35		
T14	Synchronous Ready Hold Time	0		0		
T15	HLDA Setup Time	20		20		Note 3
T16	HLDA Hold Time	10		5		Note 3
T17	Reset Setup Time	20		20		Note 3
T18	Reset Hold Time	10		10		Note 3
T19	CA Pulse Width	1 T1		1 T1		
T20	CA Setup Time	20		20		Note 3
T21	CA Hold Time	10		10		Note 3

OUTPUT TIMINGS**

Symbol	Parameter	Min	Max	Min	Max	Comments
T22	DT/R Valid Delay	0	60	0	44	
T23	$\overline{WR}$, $\overline{DEN}$ Active Delay	0	70	0	56	
T24	$\overline{WR}$, $\overline{DEN}$ Inactive Delay	10	65	10	45	
T25	Int. Active Delay	0	85	0	70	Note 4
T26	Int. Inactive Delay	0	85	0	70	Note 4
T27	Hold Active Delay	0	85	0	70	Note 4
T28	Hold Inactive Delay	0	85	0	70	Note 4
T29	Address Valid Delay	0	55	0	50	
T30	Address Float Delay	0	50	0	50	
T31	Data Valid Delay	0	55	0	50	Note 7
T32	Data Hold Time	0		0		
T33	Status Active Delay	10	60	10	45	

OUTPUT TIMINGS** (Continued)

Symbol	Parameter	82586 (8 MHz)		82586-10 (10 MHz)		Comments
		Min	Max	Min	Max	
T34	Status Inactive Delay	10	70	10	50	
T35	ALE Active Delay	0	45	0	35	Note 5
T36	ALE Inactive Delay	0	45	0	37	Note 5
T37	ALE Width	T2−10		T2−10		Note 5
T38	Address Valid to ALE Low	T2−30		T2−25		
T39	Address Hold to ALE Inactive	T4−10		T4−10		
T40	$\overline{RD}$ Active Delay	10	95	10	95	
T41	$\overline{RD}$ Inactive Delay	10	70	10	70	
T42	$\overline{RD}$ Width	2T1−50		2T1−46		
T43	Address Float to $\overline{RD}$ Active	10		0		
T44	$\overline{RD}$ Inactive to Address Active	T1−40		T1−34		
T45	$\overline{WR}$ Width	2T1−40		2T1−34		
T46	Data Hold After $\overline{WR}$	T2−25		T2−25		
T47	Control Inactive After Reset	0	60	0	60	Note 6

*All units are in ns.
**CL on all outputs is 20−200 pF unless otherwise specified.

NOTES:
1. 1.0V to 3.5V
2. 3.5V to 1.0V
3. To guarantee recognition at next clock
4. CL = 50 pF
5. CL = 100 pF
6. Affects:
   MIN MODE: $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, $\overline{DEN}$
   MAX MODE: $\overline{S0}$, $\overline{S1}$
7. High address lines (A16−A24, BHE) become valid one clock before T1 only on first memory cycle after the 82586 acquired the bus.

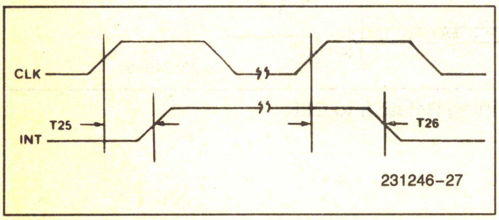


Figure 26. INT Output Timing

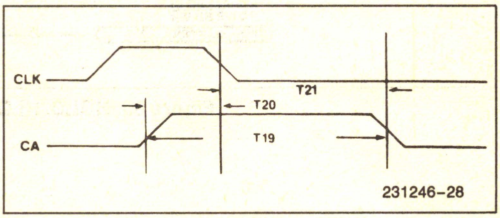


Figure 27. CA Input Timing

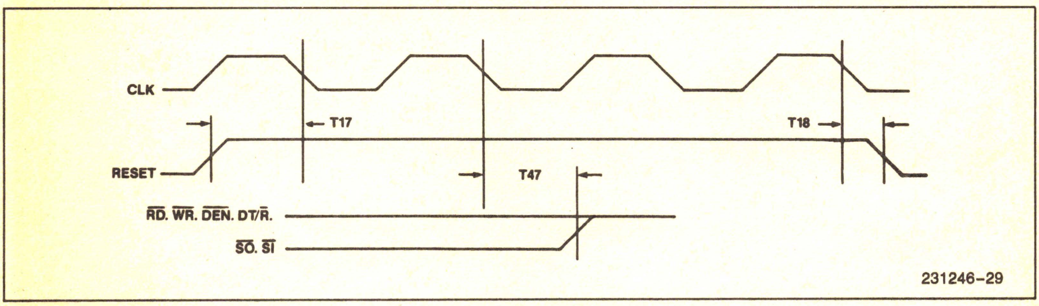


Figure 28. RESET Timing

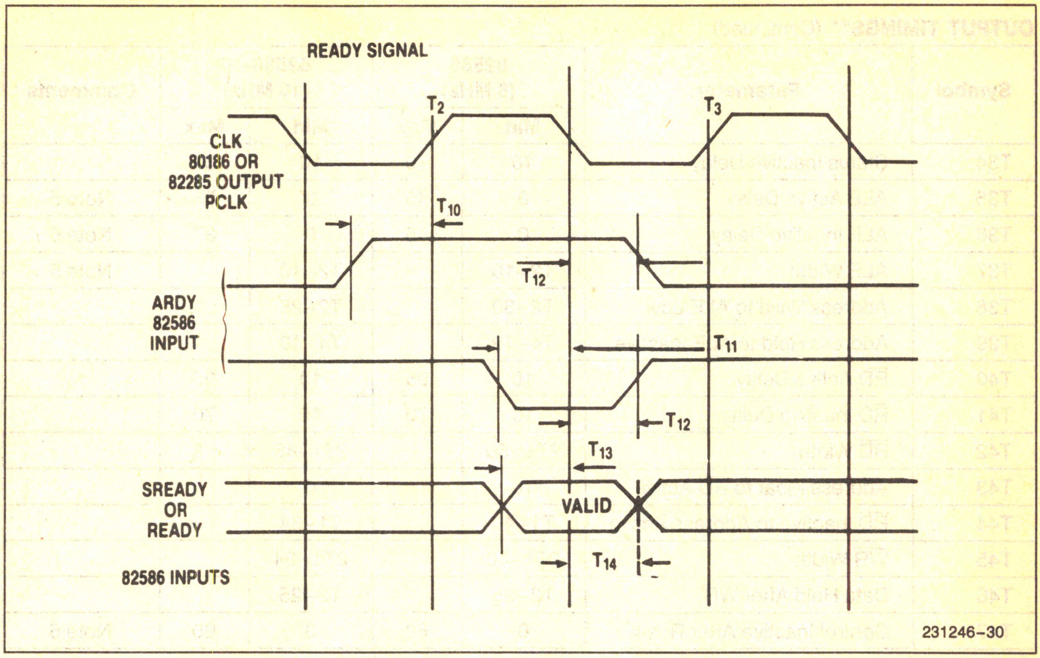


Figure 29. ARDY and SRDY Timings Relative to CLK

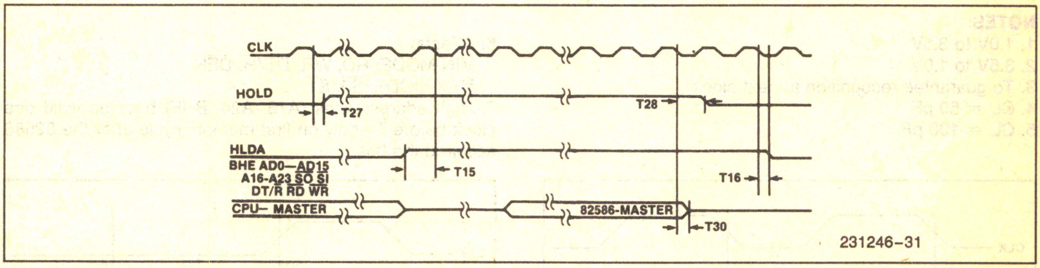


Figure 30. HOLD/HLDA Timing Relative to CLK

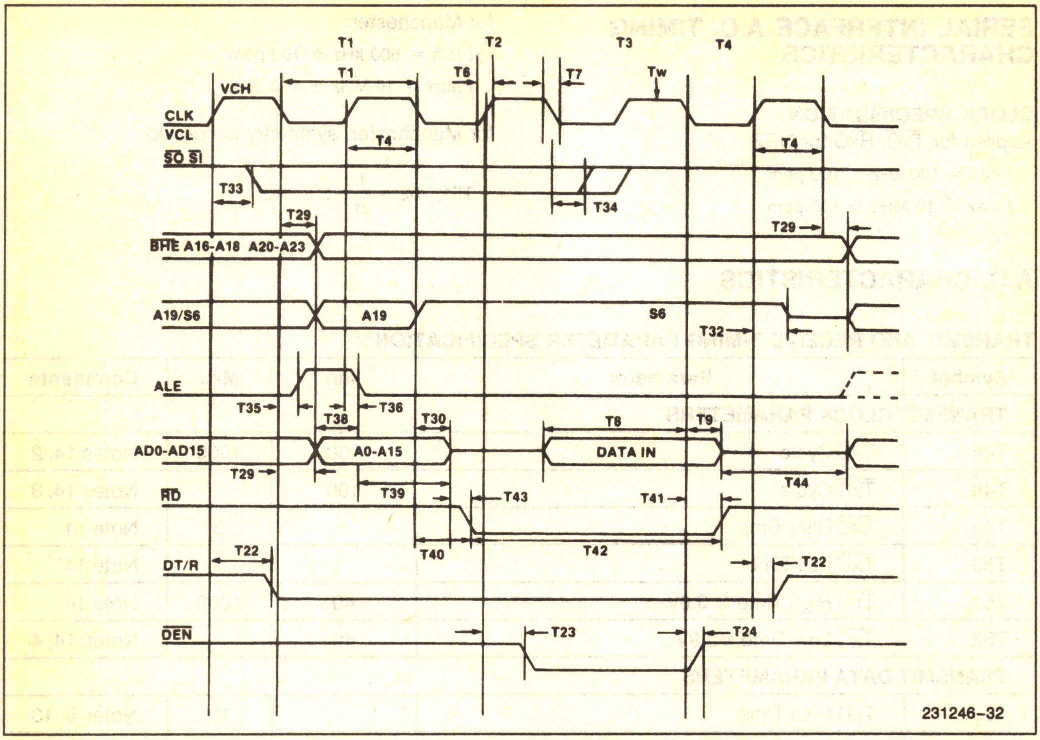


Figure 31. Read Cycle Timing

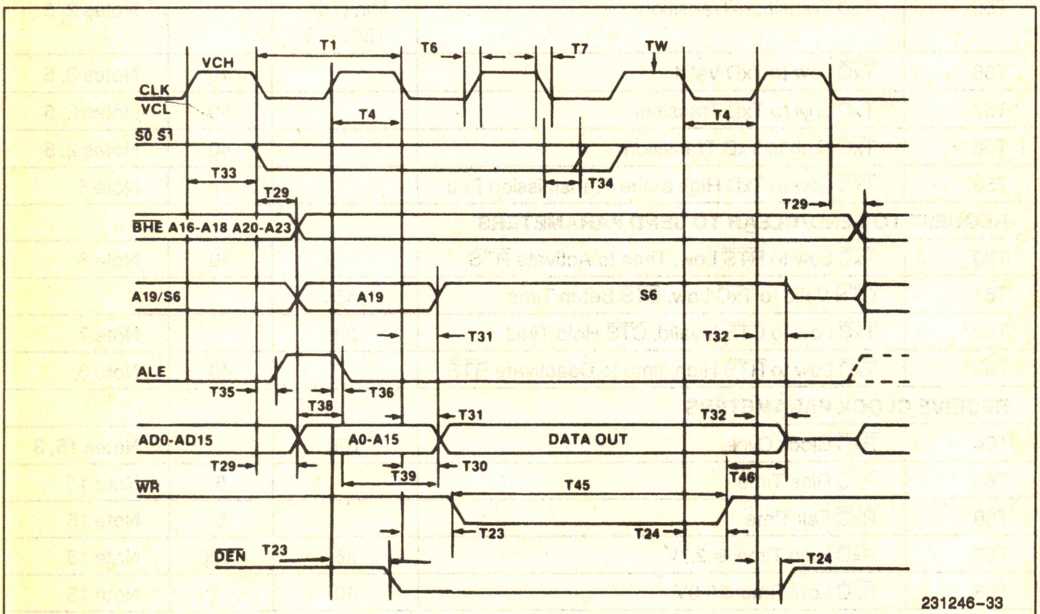


Figure 32. Write Cycle Timing

SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

CLOCK SPECIFICATION

Applies for $\overline{TxC}$, $\overline{RxC}$ for NRZ:

f min = 100 kHz ± 100 ppm

f max = 10 MHz ± 100 ppm

for Manchester:

f min = 500 kHz ± 100 ppm

f max = 10 MHz ± 100 ppm

for Manchester, symmetry is needed:

$$T51, T52 = \frac{1}{2f} \pm 5\%$$

A.C. CHARACTERISTICS

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION*

Symbol	Parameter	Min	Max	Comments
TRANSMIT CLOCK PARAMETERS				
T48	$\overline{TxC}$ Cycle	100	1000	Notes 14, 2
T48	$\overline{TxC}$ Cycle	100		Notes 14, 3
T49	$\overline{TxC}$ Rise Time		5	Note 14
T50	$\overline{TxC}$ Fall Time		5	Note 14
T51	$\overline{TxC}$ High Time @ 3.0V	40	1000	Note 14
T52	$\overline{TxC}$ Low Time @0.9V	40		Notes 14, 4
TRANSMIT DATA PARAMETERS				
T53	TxD Rise Time		10	Notes 5, 13
T54	TxD Fall Time		10	Notes 5, 13
T55	TxD Transition-Transition	Min (T51, T52) − 7		Notes 2, 5
T56	$\overline{TxC}$ Low to TxD Valid		40	Notes 3, 5
T57	$\overline{TxC}$ Low to TxD Transition		40	Notes 2, 5
T58	$\overline{TxC}$ High to TxD Transition		40	Notes 2, 5
T59	$\overline{TxC}$ Low to TxD High at the Transmission End		40	Note 5
REQUEST TO SEND/CLEAR TO SEND PARAMETERS				
T60	$\overline{TxC}$ Low to $\overline{RTS}$ Low. Time to Activate $\overline{RTS}$		40	Note 6
T61	$\overline{CTS}$ Valid to $\overline{TxC}$ Low. $\overline{CTS}$ Setup Time	45		
T62	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	20		Note 7
T63	$\overline{TxC}$ Low to $\overline{RTS}$ High, time to Deactivate $\overline{RTS}$		40	Note 6
RECEIVE CLOCK PARAMETERS				
T64	$\overline{RxC}$ Clock Cycle	100		Notes 15, 3
T65	$\overline{RxC}$ Rise Time		5	Note 15
T66	$\overline{RxC}$ Fall Time		5	Note 15
T67	$\overline{RxC}$ High Time @ 2.7V	36	1000	Note 15
T68	$\overline{RxC}$ Low Time @0.9V	40		Note 15

*All units are in ns.

A.C. CHARACTERISTICS (Continued)

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION* (Continued)

Symbol	Parameter	Min	Max	Comments
RECEIVE DATA PARAMETERS				
T69	RxD Setup Time	30		Note 1
T70	RxD Hold Time	30		Note 1
T71	RxD Rise Time		10	Note 1
T72	RxD Fall Time		10	Note 1
CARRIER SENSE/COLLISION DETECT PARAMETERS				
T73	$\overline{\text{CDT}}$ Valid to $\overline{\text{TxC}}$ High Ext. Collision Detect Setup Time	30		Note 12
T74	$\overline{\text{TxC}}$ High to $\overline{\text{CDT}}$ Inactive. $\overline{\text{CDT}}$ Hold Time	20		Note 12
T75	$\overline{\text{CDT}}$ Low to Jamming Start			Note 8
T76	$\overline{\text{CRS}}$ Valid to $\overline{\text{TxC}}$ High Ext. Carrier Sense Setup Time	30		Note 12
T77	$\overline{\text{TxC}}$ High to $\overline{\text{CRS}}$ Inactive. $\overline{\text{CRS}}$ Hold Time	20		Note 12
T78	$\overline{\text{CRS}}$ Low to Jamming Start			Note 9
T79	Jamming Period			Note 10
T80	$\overline{\text{CRS}}$ Inactive Setup Time to $\overline{\text{RxC}}$ High End of Receive Frame	60		
T81	$\overline{\text{CRS}}$ Active Hold Time from $\overline{\text{RxC}}$ High	3		
INTERFRAME SPACING PARAMETER				
T82	Inter Frame Delay			Note 11

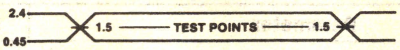
*All units are in ns.

NOTES:
1. TTL levels
2. Manchester only
3. NRZ only
4. Manchester requires 50% duty cycle
5. 1 TTL load + 50 pF
6. 1 TTL load + 100 pF
7. Abnormal end of transmission. $\overline{\text{CTS}}$ expires before $\overline{\text{RTS}}$
8. Programmable value:
   T75 = NCDF × T48 + (12.5 to 23.5) × T48 if collision occurs after preamble
   NCDF—The collision detection filter configuration value
9. Programmable value:
   T78 = NCSF × T48 + (12.5 to 23.5) × T48
   NCSF—The carrier sense filter configuration value
   TBD is a function of internal/external carrier sense bit
10. T79 = 32 × T48
11. Programmable value:
    T82 = NIFS × T48
    NIFS—the IFS configuration value

*12. To guarantee recognition on the next clock
13. Applies to TTL levels
14. 82501 compatible levels, see Figure 34
15. 82501 compatible levels, see Figure 35

A.C. TIMING CHARACTERISTICS

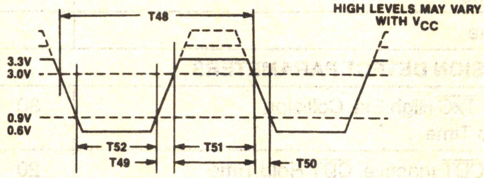
Input and Output Waveforms for AC Tests



231246-34

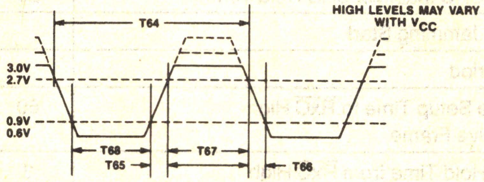
AC testing inputs are driven at 2.4V for a Logic 1 and 0.45 for a Logic 0. Timing measurements are made at 1.5V for both a Logic 1 and 0.

Figure 33. TTL Input/Output Voltage Levels for Timing Measurements



231246-35

Figure 34. $\overline{TxC}$ Input Voltage Levels for Timing Measurements



231246-36

Figure 35. $\overline{RxC}$ Input Voltage Levels for Timing Measurements

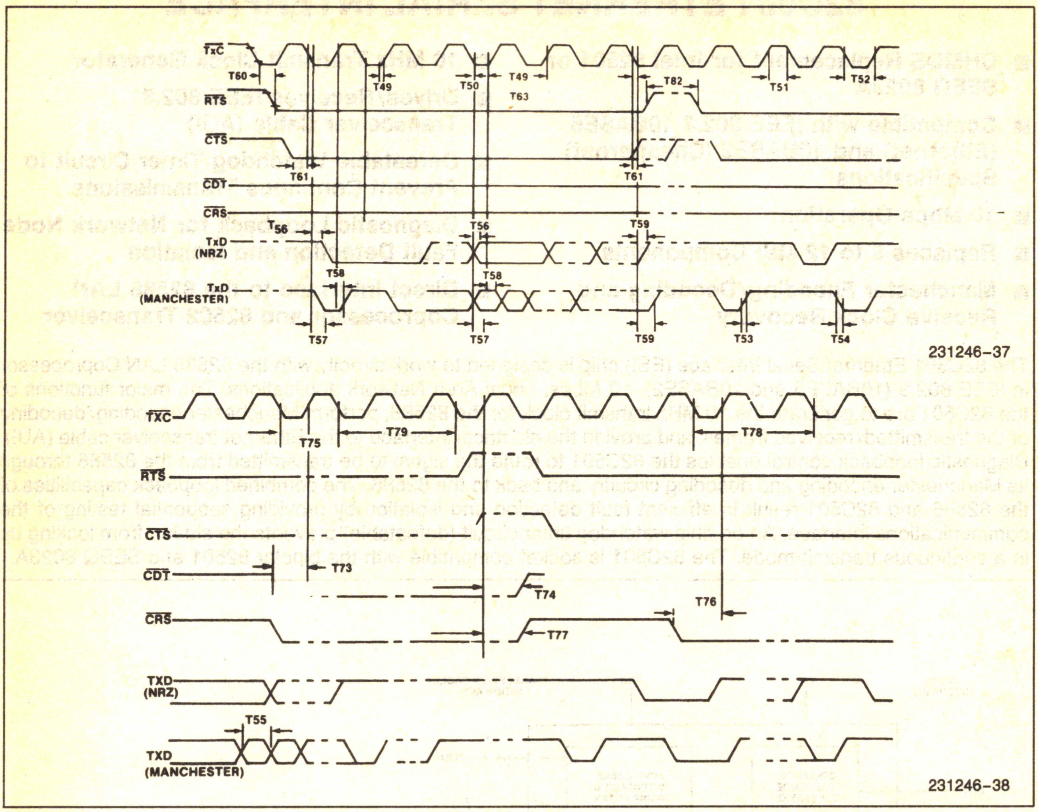


Figure 36. Transmit and Control and Data Timing

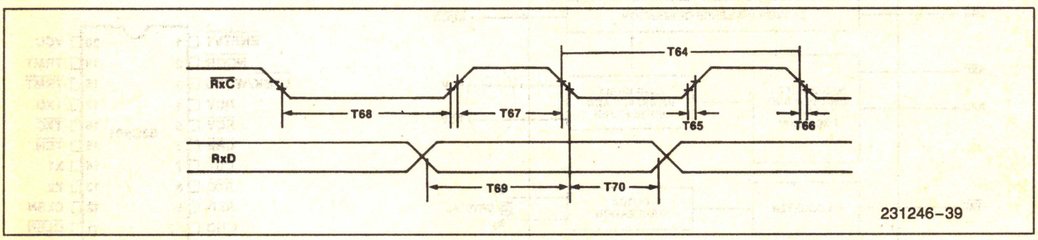


Figure 37. RxD Timing Relative to $\overline{RxC}$

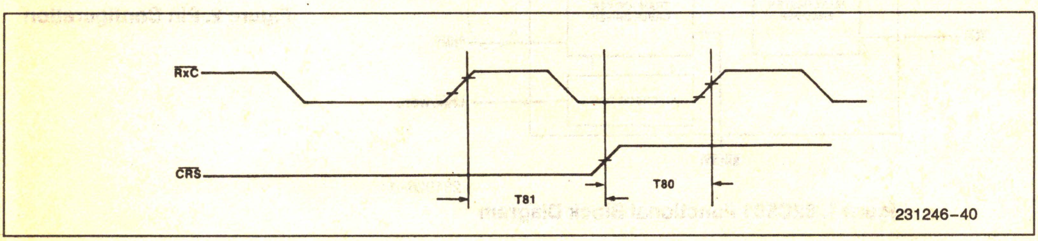
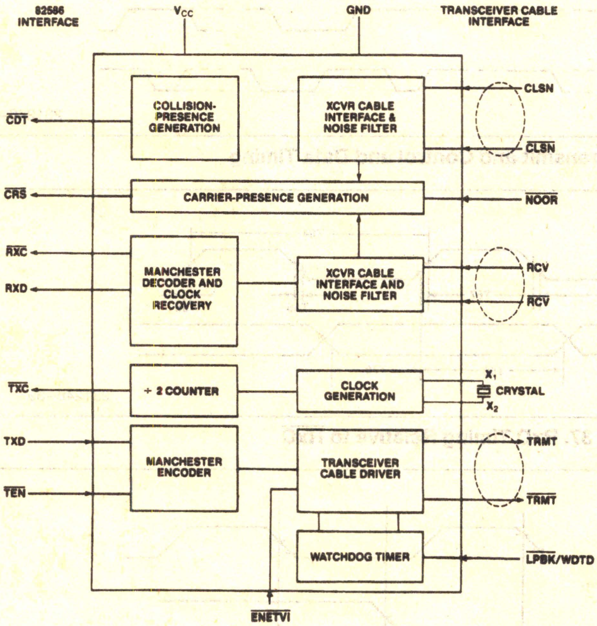


Figure 38. $\overline{CRS}$ Timing Relative to $\overline{RxC}$

82C501 ETHERNET SERIAL INTERFACE

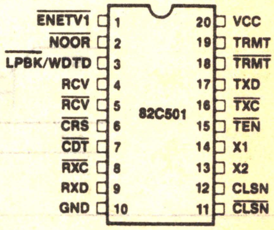
- CHMOS Replacement for Intel 82501 or SEEQ 8023A
- Compatible with IEEE 802.3 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) Specifications
- 10 Mbps Operation
- Replaces 8 to 12 MSI Components
- Manchester Encoding/Decoding and Receive Clock Recovery
- 10 MHz Transmit Clock Generator
- Drives/Receives IEEE 802.3 Transceiver Cable (AUI)
- Defeatable Watchdog Timer Circuit to Prevent Continous Transmissions
- Diagnostic Loopback for Network Node Fault Detection and Isolation
- Direct Interface to the 82586 LAN Coprocessor and 82502 Transceiver

The 82C501 Ethernet Serial Interface (ESI) chip is designed to work directly with the 82586 LAN Coprocessor in IEEE 802.3 (10BASE5 and 10BASE2), 10 Mbps, Local Area Network applications. The major functions of the 82C501 are to generate the 10 MHz transmit clock for the 82586, perform Manchester encoding/decoding of the transmitted/received frames, and provide the electrical interface to the Ethernet transceiver cable (AUI). Diagnostic loopback control enables the 82C501 to route the signal to be transmitted from the 82586 through its Manchester encoding and decoding circuitry and back to the 82586. The combined loopback capabilities of the 82586 and 82C501 result in efficient fault detection and isolation by providing sequential testing of the communications interface. An on-chip watchdog timer circuit (defeatable) prevents the station from locking up in a continuous transmit mode. The 82C501 is socket compatible with the bipolar 82501 and SEEQ 8023A.



231926-1

Figure 1. 82C501 Functional Block Diagram



231926-2

Figure 2. Pin Configuration

December 1986
Order Number: 231926-001

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{\text{ENETVI}}$	1	I	**ETHERNET VERSION 1.0:** An active low, MOS-level input. When $\overline{\text{ENETV1}}$ is asserted, the TRMT/$\overline{\text{TRMT}}$ pair remains at high differential voltage at the end of transmission. This operation is compatible with the Ethernet Version 1.0 specification. If the $\overline{\text{ENETV1}}$ pin is left floating, an internal pull-up resistor biases the input inactive high.
$\overline{\text{NOOR}}$	2	I	**CRS 'OR':** An active low, MOS-level input. When $\overline{\text{NOOR}}$ is asserted a valid signal on the collision-presence pair (CLSN/$\overline{\text{CLSN}}$) will not force $\overline{\text{CRS}}$ active low. With $\overline{\text{NOOR}}$ active $\overline{\text{CRS}}$ can only be asserted by the presence of valid data bits on the RCV/$\overline{\text{RCV}}$ pair. If the $\overline{\text{NOOR}}$ pin is floating, an internal pull-up resistor biases the input inactive high.
$\overline{\text{LPBK}}$/ WDTD	3	I	**LOOPBACK/WATCHDOG TIMER DISABLE:** An active low, TTL-level control signal enables the loopback mode. In loopback mode serial data on the TXD input is routed through the 82C501 internal circuits and back to the RXD output without driving the TRMT/$\overline{\text{TRMT}}$ output pair to the transceiver cable. During loopback $\overline{\text{CDT}}$ is asserted at the end of each transmission to simulate the SQE test. **WATCHDOG TIMER DISABLE:** An input voltage of 12V $\pm$10% through a 4 KΩ resistor will disable the on-chip watchdog timer.
RCV $\overline{\text{RCV}}$	4 5	I I	**RECEIVE PAIR:** A differentially driven input pair which is tied to the receive pair of the Ethernet transceiver cable. The first transition on RCV will be negative-going to indicate the beginning of a frame. The last transition should be positive-going to indicate the end of the frame. The received bit stream is assumed to be Manchester encoded.
$\overline{\text{CRS}}$	6	O	**CARRIER SENSE:** An active low, MOS-level output to notify the 82586 that there is activity on the coaxial cable. The signal is asserted when valid data or a collision-presence signal from the transceiver is present. It is deasserted at the end of a frame; or when the end of the collision-presence signal is detected, synchronous with $\overline{\text{RXC}}$. After transmission, once deasserted, $\overline{\text{CRS}}$ will not be reasserted again for a period of 5 μs minimum or 7 μs maximum, regardless of any activity on the collision-presence signal (CLSN/$\overline{\text{CLSN}}$) and RCV/$\overline{\text{RCV}}$ inputs.
$\overline{\text{CDT}}$	7	O	**COLLISION DETECT:** An active-low, MOS-level signal which drives the $\overline{\text{CDT}}$ input of the 82586 controller. It is asserted as long as there is activity on the collision pair (CLSN/$\overline{\text{CLSN}}$), and during SQE (heartbeat) test in loopback.
$\overline{\text{RXC}}$	8	O	**RECEIVE CLOCK:** A 10 MHz MOS level clock output with 5 ns rise and fall times. This output is connected to the 82586 receive clock input $\overline{\text{RXC}}$. There is a maximum 1.2 μs delay at the beginning of a frame reception before the clock recovery circuit gains lock. During idle (no incoming frames) $\overline{\text{RXC}}$ is forced low.
RXD	9	O	**RECEIVE DATA:** A MOS-level output tied directly to the RXD input of the 82586 controller and sampled by the 82586 at the negative edge of $\overline{\text{RXC}}$. The bit stream received from the transceiver cable is Manchester decoded prior to being transferred to the controller. This output remains high during idle.
GND	10		**GROUND:** Reference.
CLSN $\overline{\text{CLSN}}$	12 11	I I	**COLLISION PAIR:** A differentially driven input pair tied to the collision-presence pair of the Ethernet transceiver cable. The collision-presence signal is a 10 MHz square wave. The first transition at CLSN is negative-going to indicate the beginning of the signal; the last transition is positive-going to indicate the end of the signal.
X_1 X_2	14 13	I I	**CLOCK CRYSTAL:** 20 MHz crystal inputs. When X_2 is floated, X_1 can be used as an external MOS level input clock.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{TEN}$	15	I	**TRANSMIT ENABLE:** An active low, TTL level signal synchronous to $\overline{TXC}$ that enables data transmission to the transceiver cable and starts the watchdog timer. $\overline{TEN}$ can be driven by the $\overline{RTS}$ from the 82586.
$\overline{TXC}$	16	O	**TRANSMIT CLOCK:** A 10 MHz MOS level clock output with 5 ns rise and fall times. This clock is connected directly to the $\overline{TXC}$ input of the 82586.
TXD	17	I	**TRANSMIT DATA:** A TTL-level input signal that is directly connected to the serial data output, TXD, of the 82586.
TRMT $\overline{TRMT}$	19 18	O O	**TRANSMIT PAIR:** A differential output driver pair that drives the transmit pair of the transceiver cable. The output bit stream is Manchester encoded. Following the last transmission, which is always positive at TRMT, the differential voltage is slowly reduced to zero volts in a series of steps. If $\overline{ENETV1}$ is asserted this voltage stepping is disabled.
V_{CC}	20		**POWER:** 5V ±10%.

FUNCTIONAL DESCRIPTION

Clock Generation

A 20 MHz parallel resonant crystal is used to control the clock generation oscillator which provides the basic 20 MHz clock source. An internal divide-by-two counter generates the 10 MHz ± 0.01% clock required by the IEEE 802.3 specification.

It is recommended that a crystal meeting the following specifications be used:

- Quartz Crystal
- 20.00 MHz ± 0.002% @ 25°C
- Accuracy ±0.005% Over Full Operating Temperature, 0 to 70°C
- Parallel resonant with 20 pF Load Fundamental Mode

Several vendors have these crystals available; either off the shelf or custom made. Two possible vendors are:

1. M-Tron Industries, Inc
   Yankton, SD 57078

2. Crystek Corporation
   100 Crystal Drive
   Ft Myers, FL 33907

For best operation the total crystal load capacitance should be about 20 pF. Considering the internal capacitance of the 82C501, and board capacitance, a typical configuration would have a 30 to 35 pF capacitor connected between the X_1 and X_2 input and ground. Stray capacitance of the board will bring the total capacitance to 20 pF. The total length of the line on each side of the crystal (between X_1 and X_2, the crystal, and the capacitor) should be less than one inch.

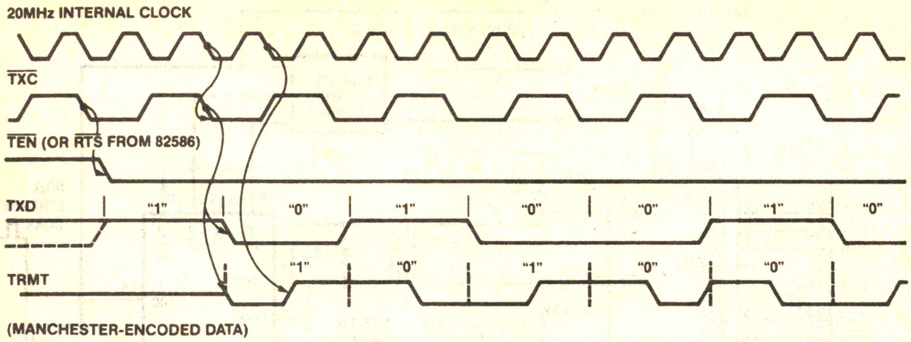
An external, 20 MHz, MOS-level clock may be applied to pin X_1 while pin X_2 is left floating.

TRANSMIT SECTION

Manchester Encoder and Transceiver Cable Driver

The 20 MHz clock is used to Manchester encode data on the TXD input line. The clock is also divided by two to produce the 10-MHz clock required by the 82586 for synchronizing its $\overline{RTS}$ and TXD signals. See Figure 3. (Note that the 82586 $\overline{RTS}$ is tied to the 82C501 $\overline{TEN}$ input as shown in Figure 4.)

Data encoding and transmission begins with $\overline{TEN}$ going low. Since the first bit is a '1', the first transition on the transmit output TRMT is always negative. Transmission ends with the $\overline{TEN}$ going high. The last transition is always poisitve at TRMT and may occur at the center of the bit cell (last bit = 1) or at the boundary of the bit cell (last bit = 0). A 1-bit delay is introduced by the 82C501 between its TXD input and TRMT/$\overline{TRMT}$ output as shown in Figure 3. If the signal applied to the $\overline{ENETV1}$ input is inactive high the TRMT output is slowly brought to its high state 200 ns after the last transmit data transition. The TRMT/$\overline{TRMT}$ differential voltage will become less than 40 mV within 8 μs after the last data transition. The undershoot for return to idle is less than 100 mV differentially. This mode of operation is compatible with the IEEE 802.3 tranceiver. See Figure 4.



231926–3

Figure 3. Start of Transmission and Manchester Encoding

If an active signal is present at the $\overline{\text{ENETV1}}$ input at the end of transmission, the TRMT/$\overline{\text{TRMT}}$ pair output will remain a high differential voltage. As a result there will be a positive differential voltage during the entire transmit idle time. This mode of operation is compatible with the Ethernet Version 1.0 specification.

Immediately after the end of a transmission all signals on the receive pair are inhibited for 5 μs minimum to 7 μs maximum. This dead time is required for proper operation of the SQE (heartbeat) test.

An internal watchdog timer is started when $\overline{\text{TEN}}$ is asserted low at the beginning of the frame. The duration of the watchdog timer is 25 ms ± 15%. If the transmission terminates (by deasserting the $\overline{\text{TEN}}$) before the timer expires, the timer is reset (and ready for the next transmission). If the timer expires before the transmission ends the frame is aborted. The frame is aborted by disabling the output driver for the TRMT/$\overline{\text{TRMT}}$ pair. RXD and $\overline{\text{RXC}}$ are not affected. The watchdog timer is reset only when the $\overline{\text{TEN}}$ is deasserted.

The cable driver is a differential circuit requiring external pulldown resistors of 240Ω ±5%. In addition, high-voltage protection of +10V maximum, and short circuit protection to ground is provided.

To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the TRMT and $\overline{\text{TRMT}}$ outputs. Transmit circuit inductance (including the IEEE 802.3 transceiver transformers) should be twenty-seven microhenrys minimum.

RECEIVE SECTION

Cable Interface and Noise Filter

The 82C501 input circuits can be driven directly from the Ethernet transceiver cable receive pair. In this case the cable is terminated with a resistor of 78Ω ± 6% for proper impedance matching. See Figure 4.

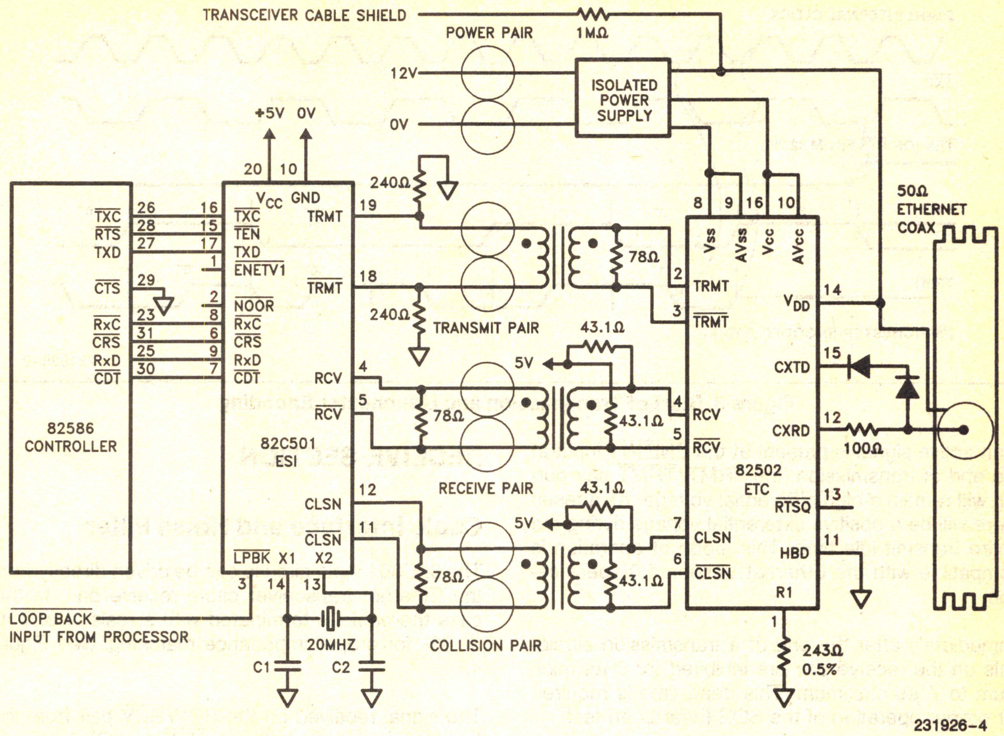
The signal received on the RCV/$\overline{\text{RCV}}$ pair from the transceiver defines both the $\overline{\text{RXC}}$ and RXD outputs to the 82586. The $\overline{\text{RXC}}$ and RXD signals are recovered from the encoded RCV/$\overline{\text{RCV}}$ pair signal by Manchester decode circuitry.

The input circuits can also be driven with ECL voltage levels. In either case, the input common mode voltage must be in the range of 0-V_{CC} volts to allow for wide driver supply variation at the transceiver. The input terminals have a 10V maximum protection. To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the RCV and $\overline{\text{RCV}}$ inputs.

A noise filter is provided at the RCV/$\overline{\text{RCV}}$ input pair to prevent spurious signals from improperly triggering the receiver circuitry. The noise filter has the following characteristics.

A negative pulse which is narrower than 10 ns, or is less than 150 mV in amplitude, is rejected during idle.

At the beginning of a reception the filter is turned off by the first negative pulse whose amplitude is greater than 275 mV and is wider than 30 ns.



NOTE:
$C_1 = C_2 = 20$ pF ±10%

Figure 4. 82586/82C501/Transceiver Cable/82502 Interface

As soon as the first valid negative pulse is recognized by the noise filter the data threshold is lowered and the $\overline{\text{CRS}}$ signal is asserted to inform the 82586 controller of the beginning of a reception. The $\overline{\text{RXC}}$ will be held low for a maximum of 1.2 μs while the synchronizer acquires a lock on the RCV signal.

If no negative transition occurs within 160 ns from the last positive transition end of frame is assumed.

After the end of a frame $\overline{\text{CRS}}$ is not inhibited. If there is a negatvie pulse whose amplitude is greater than 275 mV and is wider than 30 ns, or an equivalent sized undershoot from the transceiver, then $\overline{\text{CRS}}$ will be reactivated.

Manchester Decoder and Clock Recovery

The filtered data enters the clock recovery and decoder circuits. A Digital Bit Synchronizer (DBS) is used to extract the received clock from the data; beginning from the third negative transition of the incoming data. The DBS will acquire lock within the first twelve bit times, as seen from the RCV/$\overline{\text{RCV}}$ inputs. During that period of time the $\overline{\text{RXC}}$ is held low. The maximum, bit-cell timing distortion *(jitter)* tolerated by the DBS is ± 12 ns for the preamble, and ± 18 ns for data.

The RCV/$\overline{\text{RCV}}$ input is decoded from Manchester to NRZ and transferred synchronously with $\overline{\text{RXC}}$ to the 82586.

COLLISION-PRESENCE SECTION

The CLSN/$\overline{\text{CLSN}}$ input signal is a 10 MHz ± 15% square-wave generated by the transceiver whenever two or more data frames are superimposed on the coaxial cable. The maximum asymmetry in the CLSN/$\overline{\text{CLSN}}$ signal is 60/40% for low-to-high or

high-to-low levels. The signal is filtered for noise rejection in the same manner as RCV/$\overline{\text{RCV}}$. The noise filter rejects signals with an amplitude less than 150 mV and narrower than 10 ns. The filter turns off at the first negative pulse with an amplitude greater than 275 mV and wider than 30 ns. The filter will remain off indicating that a valid collision-presence signal is present as long as the negative CLSN/$\overline{\text{CLSN}}$ signal pulses have an amplitude greater than 275 mV. The filter returns to the 'on' state if the signal's amplitude becomes less than 150 mV, or if no negative transition occurs within 160 ns after the last positive transition. The collision filter can be immediately turned off again at the first valid signal pulse.

The common-mode voltage and external termination are identical to the RCV/$\overline{\text{RCV}}$ input. (See Figure 4.) Like the RCV/$\overline{\text{RCV}}$ input, the CLSN/$\overline{\text{CLSN}}$ input also has a 10V maximum protection and additional clamping against low-energy, high-voltage noise signals. To provide additional high voltage protection, if the cable is shorted, an isolation transformer can be used to isolate the CLSN and $\overline{\text{CLSN}}$ inputs.

A valid collision-presence signal will assert the 82C501 $\overline{\text{CDT}}$ output which can be directly tied to the $\overline{\text{CDT}}$ input of the 82586 controller. During normal operation the 82C501 logically 'OR's the collision-presence signal with internal valid data on (RCV/$\overline{\text{RCV}}$ pair) signal to generate $\overline{\text{CRS}}$ output. If, however, the $\overline{\text{NOOR}}$ input is asserted low, this 'OR' function is removed and $\overline{\text{CRS}}$ is only asserted by the presence of valid data on the RCV/$\overline{\text{RCV}}$ pair. This mode of operation is required for repeater design.

During the time that valid collision-presence transitions are present on the CLSN/$\overline{\text{CLSN}}$ input invalid data transitions may be present on the receive data pair due to the superposition of signals from two or more stations transmitting simultaneously. It is possible for RCV/$\overline{\text{RCV}}$ to lose transitions for a few bit times due to perfect cancellation of the signals; this may cause the 82C501 to abort the reception.

When a valid collision-presence signal is present the $\overline{\text{CRS}}$ signal is asserted (along with $\overline{\text{CDT}}$). However, if this collision-presence signal arrives within 5 to 7 μs from the last transmission only $\overline{\text{CDT}}$ is generated so that the 82586 recognizes the active $\overline{\text{CDT}}$ as a valid SQE (heartbeat) test signal.

Internal Loopback

When asserted, $\overline{\text{LPBK}}$ causes the 82C501 to route serial data from its TXD input through its transmit

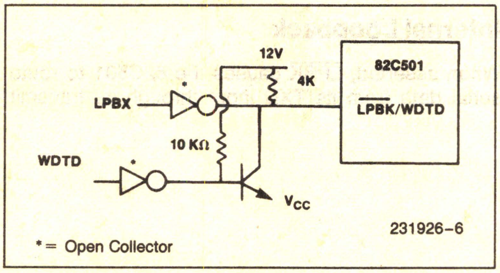
logic (retiming and Manchester encoding); returning it through the receive logic (Manchester decoding and receive clock generation) to RXD output. The internal routing prevents the data from passing through the output drivers and onto the transmit output pair TRMT/$\overline{\text{TRMT}}$. When in loopback mode all of the transmit and receive circuits, including the noise filter, are tested except for the transceiver cable output driver and input receivers. Also, at the end of each frame transmitted in loopback mode the 82C501 generates the SQE test (heartbeat) signal within 1 μs after the end of the frame. Thus, the collsion circuits, including the noise filter, are also tested in loopback mode.

The watchdog timer remains enabled in loopback mode, terminating test frames that exceed its timeout period. The watchdog can be inhibited by connecting $\overline{\text{LPBK}}$ to a 4 kΩ resistor connected to 12V $\pm$10%. The loopback feature can still be used to test the integrity of the 82C501 by using the circuit shown in Figure 5.

The 82C501 operates as a full-duplex device, being able to transmit and receive simultaneously. By combining the internal and external loopback modes of the 82586, and the internal loopback and normal modes of the 82C501, incremental testing of an 82586/82C501-based interface can be performed under program control for systematic fault detection and fault isolation.

Interface Example

The 82C501 is designed to work directly with the 82586 controller in IEEE 802.3 10 Mbps, as well as other 10 Mbps LAN applications. The control and data signals connect directly between the two devices without the need for additional external logic. The complete 82586/82C501 Ethernet Transceiver cable/82502 interface is shown in Figure 4. The 82C501 provides the driver and receivers needed to directly connect to the transceiver cable; requiring only terminating resistors on each input signal pair and 240Ω pull-down resistors.



LPBK	WDTD	Function
1	X	LPBK mode
0	0	Normal mode
0	*1	Normal mode with watchdog timer disabled

Figure 5. Watchdog Timer Disable

ABSOLUTE MAXIMUM RATING

Ambient Temperature Under Bias .... 0°C to +70°C
Case Temperature Under Bias .... 0°C to +90°C
Storage Temperature .... −65°C to +140°C
All Output and Supply Voltages ..... −0.5V to +7V
All Input Voltages .... −1.0V to +6.0V(1)
Power Dissipation 0.5W

**Notice:* Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $T_C = 0°C$ to $+90°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter		Min	Max
V_{IL}	Input Low Voltage	TTL MOS	−0.5V −0.5V	0.8V 0.6V
V_{IH}	Input High Voltage	TTL MOS	2.0V 3.9V	V_{CC} + 0.5V V_{CC} + 0.5V
V_{IDF}	Input Differential Voltage		±300 mV	±1500 mV
V_{CM}	Input Common Mode Voltage		0V	V_{CC}
V_{OCM}	Common Mode Output(2)		0.5V	5.0V
V_{OL}	Output Low Voltage @ I_{OL} = 4 mA			0.45V
V_{OH}	Output High Voltage (MOS) @ I_{OH} = −500 μA		3.9V	
V_{ODF}	Differential Output Swing(3)		±0.55V	±1.2V
I_{LI}	Input Leakage Current(4) @ V_{IN} = 0V to V_{CC}			±50 μA
C_{IN}	Input Capacitance @ fc = 1 MHz			10 pF
C_{OUT}	Output Capacitance @ fc = 1 MHz			20 pF
I_{CC}	Power Supply Current @ T_A = 70°C(5)			100 mA
V_{REJECT}	Differential Reject Voltage			150 mV
V_{ACCEPT}	Differential Accept Voltage		275 mV	

NOTES:
1. The voltage levels for CLSN/$\overline{\text{CLSN}}$, RCV/$\overline{\text{RCV}}$ inputs are −1.5V to +10V.
2. The load is a 78Ω ±6% resistor in parallel with a 27 μH ±1% inductor.
3. DC measurement values.
4. Only TXD, $\overline{\text{TEN}}$, and X_1 conform to this spec. $\overline{\text{NOOR}}$, $\overline{\text{ENETV1}}$, and $\overline{\text{LPBK}}$/WDTD inputs source/sink a maximum of 1 mA.
5. Part of the power is dissipated through the pulldown resistors connected to TRMT/$\overline{\text{TRMT}}$ outputs.

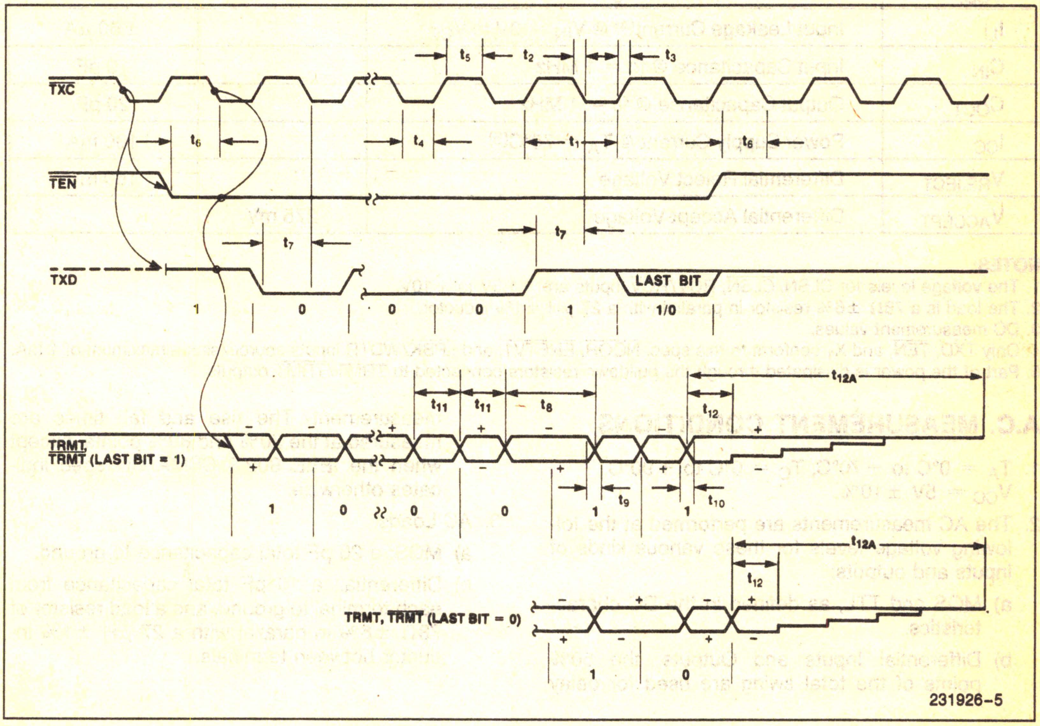
A.C. MEASUREMENT CONDITIONS

1. $T_A = 0°C$ to $+70°C$, $T_C = 0°C$ to $+90°C$, $V_{CC} = 5V \pm 10\%$.
2. The AC measurements are performed at the following voltage levels for these various kinds of inputs and outputs:
   a) MOS and TTL, as defined in the DC characteristics.
   b) Differential Inputs and Outputs: the 50% points of the total swing are used for delay measurement. The rise and fall times are measured at the 20% and 80% points, except when the IEEE 802.3 CSMA/CD spec indicates otherwise.
3. AC Loads:
   a) MOS: a 20 pF total capacitance to ground.
   b) Differential: a 10 pF total capacitance from each terminal to ground, and a load resistor of 78Ω ±6% in parallel with a 27 μH ±1% inductor between terminals.

TRANSMIT TIMING(3)

Symbol	Parameter	Min	Max	Unit
t_1	$\overline{TXC}$ Cycle Time	99.99	100.01	ns
t_2	$\overline{TXC}$ Fall Time(1)		5	ns
t_3	$\overline{TXC}$ Rise Time(1)		5	ns
t_4	$\overline{TXC}$ Low Time (at 0.9V)	40		ns
t_5	$\overline{TXC}$ High Time (at 3.0V)	40		ns
t_6	Transmit Enable/Disable to $\overline{TXC}$ Low	45		ns
t_7	TXD Stable to $\overline{TXC}$ Low	45		ns
t_8	Bit Cell Center to Bit Cell Center of Transmit Pair Data	99.5	100.5	ns
t_9	Transmit Pair Data Fall Time(2)	1.0	5.0	ns
t_{10}	Transmit Pair Data Rise Time(2)	1.0	5.0	ns
t_{11}	Bit Cell Center to Bit Cell Boundary of Transmit Pair Data	49.5	50.5	ns
t_{12}	$\overline{TRMT}$ held low from Last Positive Transition of Transmit Pair Data during idle:	200		ns
t_{12A}	From Last Positive Transition of Transmit Pair Differential Output Approaches Within 40 mV of zero volts.		8000	ns

NOTES:
1. Rise and fall times are guaranteed by design.
2. Measured per 802.3 Para. 6.5.1.1.
3. t_{22}, t_{23}, t_{24}, t_{39} do not appear.



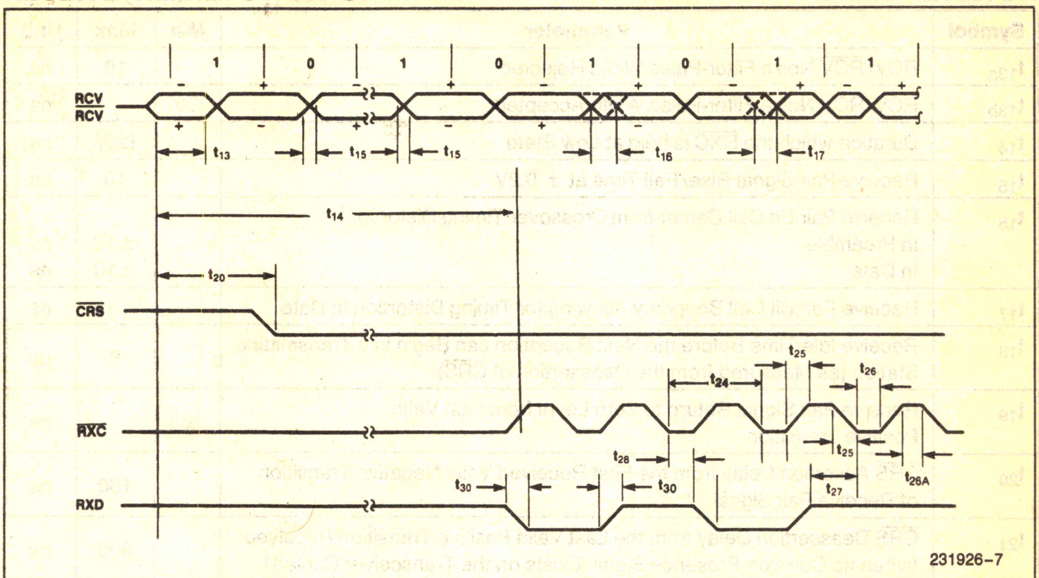
231926–5

RECEIVE TIMING

Symbol	Parameter	Min	Max	Unit
t_{13a}	RCV/$\overline{\text{RCV}}$ Noise Filter-Pulse Width Rejected		10	ns
t_{13b}	RCV/$\overline{\text{RCV}}$ Noise Filter-Pulse Width Accepted	30		ns
t_{14}	Duration which the $\overline{\text{RXC}}$ is held at Low State		1200	ns
t_{15}	Receive Pair Signal Rise/Fall Time at ± 0.2V		10	ns
t_{16}	Receive Pair Bit Cell Center from Crossover Timing Distortion: In Preamble In Data		 ±12 ±18	 ns ns
t_{17}	Receive Pair Bit Cell Boundary Allowing for Timing Distortion In Data		±18	ns
t_{18}	Receive Idle Time Before the Next Reception can Begin in a Transmitting Station (as Measured from the Deassertion of $\overline{\text{CRS}}$)		8	μs
t_{19}	Receive Pair Signal Return to Zero Level from Last Valid Positive Transition	160		ns
t_{20}	$\overline{\text{CRS}}$ Assertion Delay from the First Received Valid Negative Transition of Receive Pair Signal		100	ns
t_{21}	$\overline{\text{CRS}}$ Deassertion Delay from the Last Valid Positive Transition Received (when no Collision-Presence Signal Exists on the Transceiver Cable)(1)		300	ns
t_{25}	$\overline{\text{RXC}}$ Rise/Fall Time(2)		5.0	ns
t_{26}	$\overline{\text{RXC}}$ Low Time (at 0.9V)	40		ns
t_{26A}	$\overline{\text{RXC}}$ High Time (at 3.0V)	36		ns
t_{27}	Receive Data Stable Before the Negative Edge of $\overline{\text{RXC}}$	30		ns
t_{28}	Receive Data Held Valid Past the Negative Edge of $\overline{\text{RXC}}$	30		ns
t_{29}	Carrier Sense Active Hold Time from $\overline{\text{RXC}}$ High	10	40	ns
t_{30}	Receive Data Rise/Fall Time		10	ns
t_{31}	$\overline{\text{CRS}}$ Inhibit Time After Frame Transmission(3)	5	7	μs

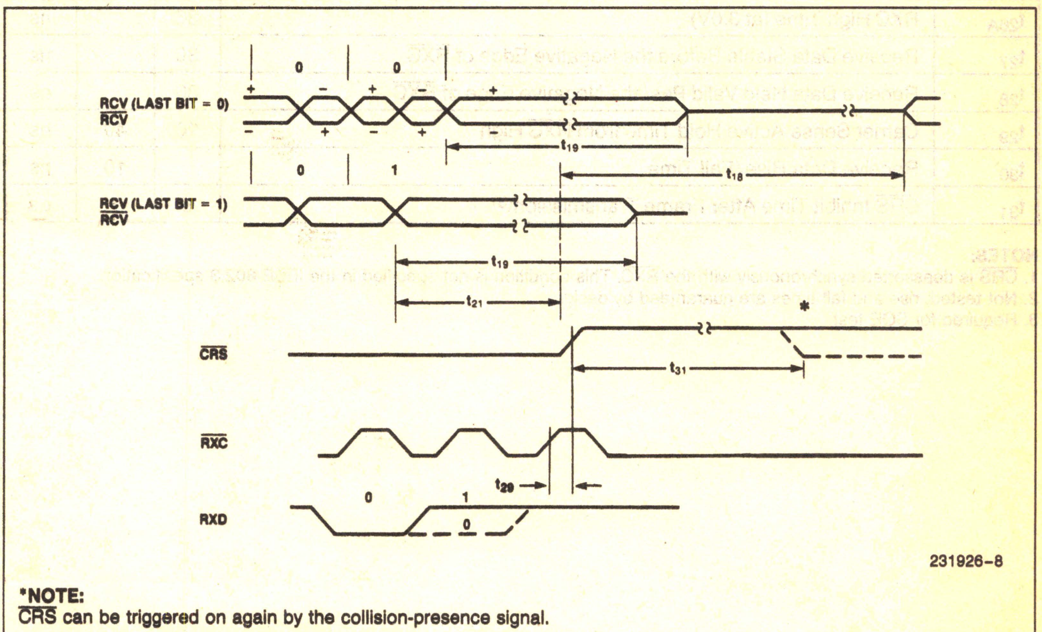
NOTES:
1. $\overline{\text{CRS}}$ is deasserted synchronously with the $\overline{\text{RXC}}$. This condition is not specified in the IEEE 802.3 specification.
2. Not tested, rise and fall times are guaranteed by design.
3. Required for SQE test.

RECEIVE TIMING: START OF FRAME



231926-7

RECEIVE TIMING: END OF FRAME

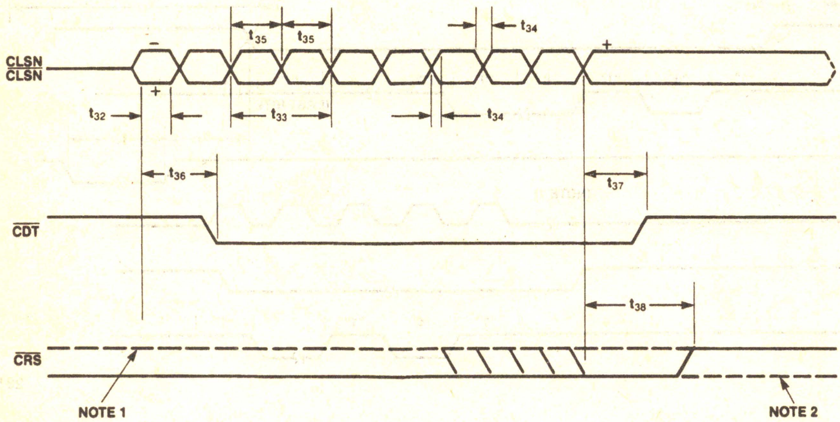


231926-8

***NOTE:**
CRS can be triggered on again by the collision-presence signal.

COLLISION TIMING

Symbol	Parameter	Min	Max	Unit
t_{32a}	CLSN/$\overline{\text{CLSN}}$ Noise Filter-Pulse Width Rejected		10	ns
t_{32b}	CLSN/$\overline{\text{CLSN}}$ Noise Filter-Pulse Width Accepted	30		ns
t_{33}	CLSN/$\overline{\text{CLSN}}$ Cycle Time	86	118	ns
t_{34}	CLSN/$\overline{\text{CLSN}}$ Rise/Fall Time at ± 0.2V		15	ns
t_{35}	CLSN/$\overline{\text{CLSN}}$ Transition Time	35	70	ns
t_{36}	$\overline{\text{CDT}}$ Assertion from the First Valid Negative Edge of Collision Pair Signal		75	ns
t_{37}	$\overline{\text{CDT}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ Signal		300	ns
t_{38}	$\overline{\text{CRS}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ Signal (if no Post-Collision Signal Remains on the Receive Pair)		450	ns



231926-9

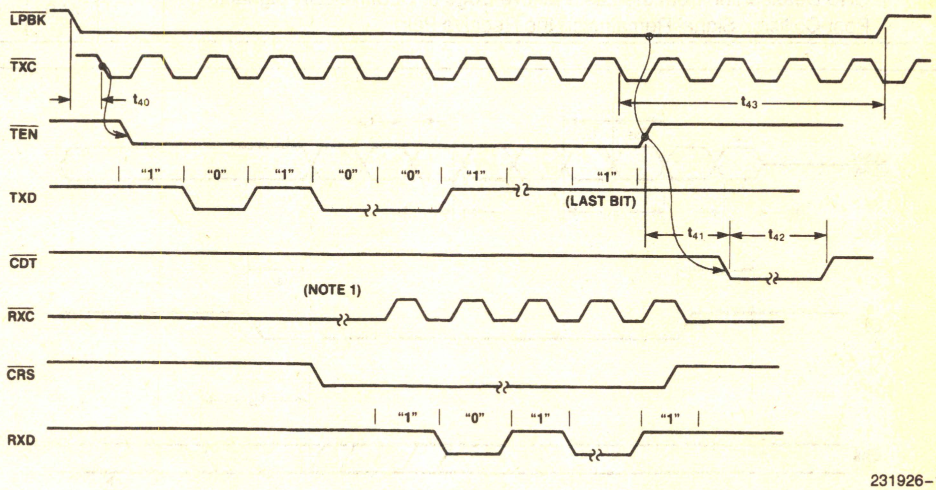
NOTES:

1. $\overline{\text{CRS}}$ will be deasserted for a period up to 7 μs maximum when RCV/$\overline{\text{RCV}}$ or CLSN/$\overline{\text{CLSN}}$ terminates, whichever occurs later.
2. $\overline{\text{CRS}}$ will remain asserted after the CLSN/$\overline{\text{CLSN}}$ signal terminates if RCV/$\overline{\text{RCV}}$ signals continue.

LOOPBACK TIMING

Symbol	Parameter	Min	Max	Unit
t_{40}	$\overline{LPBK}$ asserted before the first attempted transmission (1)	500		ns
t_{41}	Simulated collision test delay from the end of each attempted transmission	0.5	1.5	μs
t_{42}	Simulated collision test duration(2)	0.5	1.0	μs
t_{43}	$\overline{LPBK}$ deasserted after the last attempted transmission	5		μs

NOTES:
1. In Loopback mode, $\overline{RXC}$, $\overline{RXC}$ and $\overline{CRS}$ function in the same manner as a normal Receive.
2. SQE test (heartbeat) signal



231926-10

NOTE:
1. During Loopback, the 82C501 receive circuitry uses 12 bit times while the Digital Bit Synchronizer locks on the data. As a result, the first 12 bits are lost and $\overline{RXC}$ is held low during that time.

TESTABILITY

NOTE:
1. All AC Parameters become valid after the Digital Bit Synchronizer has stabilized: 100 μs after the application of power.

82502
ETHERNET TRANSCEIVER CHIP

- **Conforms to Following Standards:**
  - **— IEEE 802.3, 10BASE5 (Ethernet)**
  - **— IEEE 802.3, 10BASE2 (Cheapernet)**
  - **— Ethernet Version 2.0**
- **Jabber Function**
- **Receive Based Collision Detection**
- **Defeatable Signal Quality Error (Heartbeat) Test**
- **Requires Minimum Board Space**
  - **— On-Chip Voltage Reference**
  - **— 16 Pin DIP**
- **No External Adjustments Required**
- **Reliable CHMOS Technology**

The 82502 Ethernet Transceiver Chip is a CHMOS LSI device that provides the complete set of transmit, receive and collision detection functions specified by the IEEE 802.3, 10BASE5 (Ethernet) and 10BASE2 (Cheapernet) 10 Mbps baseband standards for the Media Attachment Unit (MAU). The 82502 teams up with Intel's 82586 LAN Coprocessor and the 82501 Ethernet Serial Interface enabling the designer to implement highly integrated IEEE 802.3 systems.

Three basic functional blocks make up the 82502: transmit, receive and collision detection. The transmit and receive sections transfer data from the transceiver drop (Access Unit Interface or AUI) cable to the coaxial cable of the network and vice-versa. The collision detection section senses simultaneous transmissions by two or more network stations (collisions) on the coaxial cable and reacts by sending a 10 MHz signal across the transceiver drop cable to the station that it front ends.

When used in an Ethernet application, the 82502 can drive a transceiver cable up to 50 meters in length (for Cheapernet, there is no transceiver cable). The 82502 provides all active communications circuitry for the transceiver function in the Ethernet/Cheapernet environment. It is an ideal companion to the Intel 82501.

The 82502 is fabricated in Intel's reliable 10V CHMOS II process.

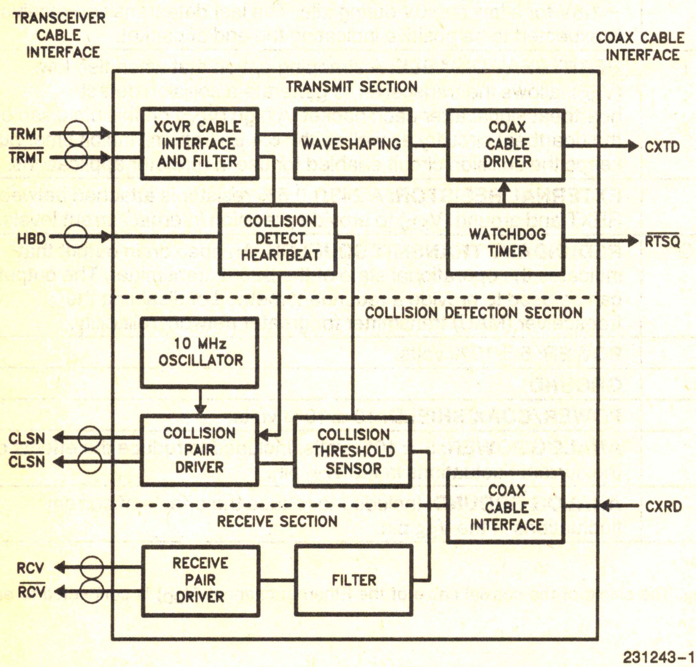


Figure 1. 82502 Functional Block Diagram

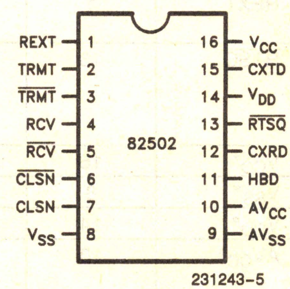


Figure 2. 82502 Pin Diagram

October 1986
Order Number: 231243-004

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
TRMT, $\overline{\text{TRMT}}$	2 3	I I	**TRANSMIT DATA PAIR:** A differentially driven input tied to the transmit pair of the transceiver cable. The transmit pair of the transceiver cable is driven with 10 Mbps Manchester encoded data from the serial interface of the data link (82501). TRMT/$\overline{\text{TRMT}}$ must be isolated from the transceiver cable by a pulse transformer. The last transition is expected to be positive indicating end of packet.
RCV, $\overline{\text{RCV}}$	4 5	O O	**RECEIVE DATA PAIR:** An output driver pair that generates an ECL AC signal level to drive the transceiver cable receive pair with the 10 Mbps Manchester encoded data received from the coaxial cable of the network. RCV/$\overline{\text{RCV}}$ must be isolated from the transceiver cable by a pulse transformer. The last transition is always positive indicating the end of the packet. The current from the $\overline{\text{RCV}}$ pin is incrementally decreased after the last transition.
CLSN, $\overline{\text{CLSN}}$	7 6	O O	**COLLISION PRESENCE PAIR:** An output driver pair that generates a 10 MHz ECL AC signal level square wave on the collision presence pair of the transceiver cable when: a collision is detected on the coaxial cable of the network, during self-test as the collision circuit heartbeat indication, or after the watchdog timer has expired to indicate that the coaxial cable transmitter is disabled.
CXTD	15	O	**COAXIAL CABLE TRANSMIT DATA:** An output pin that transmits data onto the coaxial cable of the network by sinking current from the center conductor of the coaxial cable. The last data transition at the end of a packet is always low to high.
CXRD	12	I	**COAXIAL CABLE RECEIVE DATA:** An input pin that receives data from the coaxial cable of the network. Typical signal levels (referenced to V_{DD}) received on CXRD are −200 mV for high, −1.8V for a low and 0V during idle. The last data transition received is expected to be positive indicating the end of packet.
HBD	11	I	**HEARTBEAT DISABLE:** A strapping option that when tied low (V_{SS}), allows the transceiver to generate a collision detect heartbeat signal after each packet. A high (V_{CC}) on this pin disables the heartbeat circuitry as well as the 6.4 μs transmit inhibit timer but keeps the collision circuit enabled for use in repeater applications.
REXT	1		**EXTERNAL RESISTOR:** A 243Ω 0.5% resistor is attached between REXT and ground (V_{SS}) to provide precision internal current levels.
$\overline{\text{RTSQ}}$	13	O	**REDUNDANT TRANSMIT SQUELCH:** An open drain output that indicates the operational state of the 82502 transmitter. The output can be used to provide a redundant method of disabling the transceiver (MAU) transmitter for greater network reliability.
V_{CC}*	16		**POWER:** 5 ±10% volts.
V_{SS}*	8		**GROUND**
V_{DD}*	14		**POWER/COAX SHIELD:** 10 ±10% volts.
AV_{CC}*	10		**ANALOG POWER:** 5 ±10% volts. Included to reduce the effects of the current fluctuations in the V_{CC} pin.
AV_{SS}*	9		**ANALOG GROUND:** Included to reduce the effects of current fluctuations in the V_{SS} pin.

***NOTE:**
These voltages are referenced to V_{SS}. The shield of the coaxial cable of the Ethernet channel (V_{DD}) is connected to earth ground.

FUNCTIONAL DESCRIPTION

Transmit Section

Transmit Pair Receiver

The 82502 receives transmit data from an Ethernet Serial Interface (ESI) on the TRMT pins and transmits it onto a 50Ω coax cable (the Ethernet channel). The TRMT pins are connected to the transmit pair of a transceiver cable through a pulse transformer (see Figure 3). The transformer provides the isolation required between the transceiver cable and the coax cable. The DC bias for the TRMT pins is provided internally. The 78 ±5% Ω resistor for the TRMT input termination is external to the device.

A noise filter (squelch) is provided at the TRMT input pair to prevent spurious noise signals received on the pins from being transmitted onto the coaxial cable. The squelch is on during idle and rejects signals not more negative than −160 mV and/or less than 16 ns wide. The squelch is removed if the signal on the TRMT pins becomes more negative than −225 mV for 42 ns or more.

Once the squelch is removed, the transmit signal received on the device's TRMT pins measured differentially may be between ±0.25V minimum and ±1.2V maximum. The squelch will remain off if the received signal is not more positive than −225 mV or if it is more positive for less than 111 ns. The squelch will turn back on if the received signal is more positive than −160 mV for more than 215 ns. When the squelch turns back on a non-retriggerable 6.4 μs transmit inhibit timer is started which disables the transmitter within 250 ns after the last transition for a minimum of 5.5 μs and a maximum of 8 μs.

The transmit data path through the device consists of a receiver, rise and fall time control circuitry, and coaxial cable current driver. The maximum timing distortion introduced by the device's transmit path is ±2 ns per data transition edge (t3) including timing distortion caused by output rise and fall time difference. The maximum steady-state propagation delay of the transmit path is 50 ns.

Coaxial Cable Driver

The CXTD pin transmits data onto the coaxial cable by sinking current from the center conductor of the coaxial cable. The DC component of the output current level is between 37 mA and 45 mA. The AC component is from ±28 mA up to the DC component value. The idle current level is less than 5 μA. Positive current is defined as current into the device. The output resistance is 9 KΩ minimum.

At the start of a frame transmission, no more than 2 bits of the frame information may be received from the TRMT pair and not transmitted onto the coaxial cable. The first bit transmitted by the 82502 may contain phase violations, however all successive bits of the frame will be valid.

Watchdog Timer (Jabber Function)

The 82502 Watchdog Timer will inhibit continuous transmissions onto the coaxial cable of duration greater than 52 ms ±18%. The timer starts timing when the TRMT squelch is turned off to receive valid transmit data on the TRMT pins. If a frame transmission is completed within the 52 ms time slot, the timer is reset 2 μs to 6 μs after the last bit of the frame has been transmitted. Two to six microseconds of reset delay is provided so that a short absence of transmit data will not reset the watchdog timer. If a new transmission is started within the reset delay time, the timer will not be reset.

If the timer times out (i.e., transmit frame duration exceeds 52 ms ±18%) the transmit section is disabled and the active collision signal is transmitted out onto the collision pair until transmit data received on the TRMT pair is discontinued. When the TRMT inputs become idle, the collision signal is turned off and a 420 ms (typical) non-retriggerable timer is started. During this 420 ms, the transmit section is disabled. The transmit section is re-enabled after 420 ms.

Redundant Transmit Squelch Output

The $\overline{\text{RTSQ}}$ pin is an open drain output which indicates the operational state of the 82502 transmit section. The output with a pull-up resistor becomes high when the transmit squelch is removed and a valid transmission started. The output remains high as long as the transmission continues onto the cable. When the transmission is successfully completed or terminated by a watchdog timer timeout, the output becomes low. An application of this pin, for example, is a redundant transmit squelch circuit shown in Figure 4. Two 82502 devices are used for fault tolerance. The second 82502 monitors the TRMT pair of the first 82502, which is the primary transceiver device. When the $\overline{\text{RTSQ}}$ of the second 82502 is low (i.e., the TRMT pair is idle), the CXTD pin of the first 82502 is disabled by the PNP transistor. The PNP transistor is in saturation since the $\overline{\text{RTSQ}}$ pin is sinking large current through the voltage divider. When the $\overline{\text{RTSQ}}$ is high, the CXTD pin of the first 82502 can transmit a frame onto the cable.

Receive Section

Coaxial Cable Receiver

The 82502's coaxial cable receive pins "V_{DD}" and "CXRD" are connected to the coax cable shield and center conductor, respectively.

The shunt capacitance measured between the V_{DD} and CXRD pins is typically 2.0 pF. The shunt resistance is greater than 1 MΩ. The input leakage current on the CXRD pin is between −1 μA and 20 μA. These conditions apply in both the power off and power on states.

A noise filter is provided at the V_{DD} and CXRD inputs to reject noise signals during idle. Signals received on the coax whose DC component (frequency less than 1 MHz) does not drop below −120 mV will be rejected. The noise filter will be disabled if the signal's DC component drops below −350 mV. A maximum of 4 bits will be received from the coaxial cable and may not be transmitted onto the RCV pair. The first bit transmitted may contain phase violations, however all successive bits of the frame will be valid.

The coaxial cable receiver compensation circuit is provided such that the combination of the coaxial cable and the receive circuit do not result in more than ±6 ns of edge jitter (t10) for untapped minimum and maximum length Ethernet cables in a noise-free environment.

The noise filter will remain off if the received signal's AC component is not positive for more than 136 ns, and the DC component remains below −350 mV. While the filter is off, the received data will be transmitted onto the RCV pair with less than 50 ns steady state propagation delay. The timing distortion on the RCV pair output is less than ±2 ns per edge for a 400 mV peak-to-peak 5 MHz sinusoidal input from the coaxial cable.

The noise filter will turn back on if the received signal's AC component is positive for more than 210 ns, or if the DC component is more positive than −120 mV. The RXD driver is disabled within 275 ns after the last received data transition.

Receive Pair Driver

The RCV outputs are connected to the receive pair of a transceiver cable through a pulse transformer.

The receive pair (RXD/$\overline{\text{RXD}}$) transceiver cable driver is a current driver. The output pins require 43.2Ω ±1% pull-up resistors to V_{CC}. The differential output voltage level is between ±580 mV minimum and ±1V maximum, for a combined differential inductive load of 25 μH minimum and either 73Ω or 83Ω ±1% resistive load.

Between 200 ns and 325 ns after the last output data transition on the RCV/$\overline{\text{RCV}}$ pins, the 82502 begins incrementally decreasing the $\overline{\text{RCV}}$ output current, using a series of small steps until it becomes idle. The maximum idle current levels on the RCV and $\overline{\text{RCV}}$ pins are 0.2 mA and 7 mA, respectively.

The purpose of the stepping sequence is to guarantee that the RCV/$\overline{\text{RCV}}$ differential output signal (measured with a 78 ±5% terminating resistor in parallel with a minimum magnetizing inductance of 25 μH) will remain high for a minimum of 200 ns following the last output transition, will not undershoot more than 80 mV and will limit the current stored in the inductor to 4 mA after 80 bit times.

Collision Detect Section

The collision presence outputs are connected to the collision pair of a transceiver through a pulse transformer. The collision presence outputs are used by the device to indicate one of three conditions: (1) there are simultaneous transmission attempts by two or more stations on the coax, (2) self test after a transmission proving that the collision circuitry is functional or (3) a frame transmission time exceeds the 52 ms watchdog timer time limit. The active collision presence signals is a 10 MHz ±15% ECL level square wave transmitted on the collision pair of the transceiver cable. The duty cycle of the square wave is better than 40/60 or 60/40. The collision pair driver has the same output signal characteristics as the receive pair driver.

The collision threshold DC level set inside the 82502 corresponds to −1.495 ±5% volts on the coaxial cable. When the collision filter's impulse response is taken into account, the threshold corresponds to a collision threshold between −1.492V and −1.58V. If the DC average signal level detected on the coaxial cable is more negative than the threshold, the on-chip oscillator transmits an active collision presence signal onto the collision pair. The delay in the device from the moment it detects a collision on the CXRD pin to the collision presence signal transmission onto the collision pair is less than 0.9 μs. With a 500 meter coaxial cable, it is less than 2.9 μs including the propagation delay of the coaxial cable.

Self Test

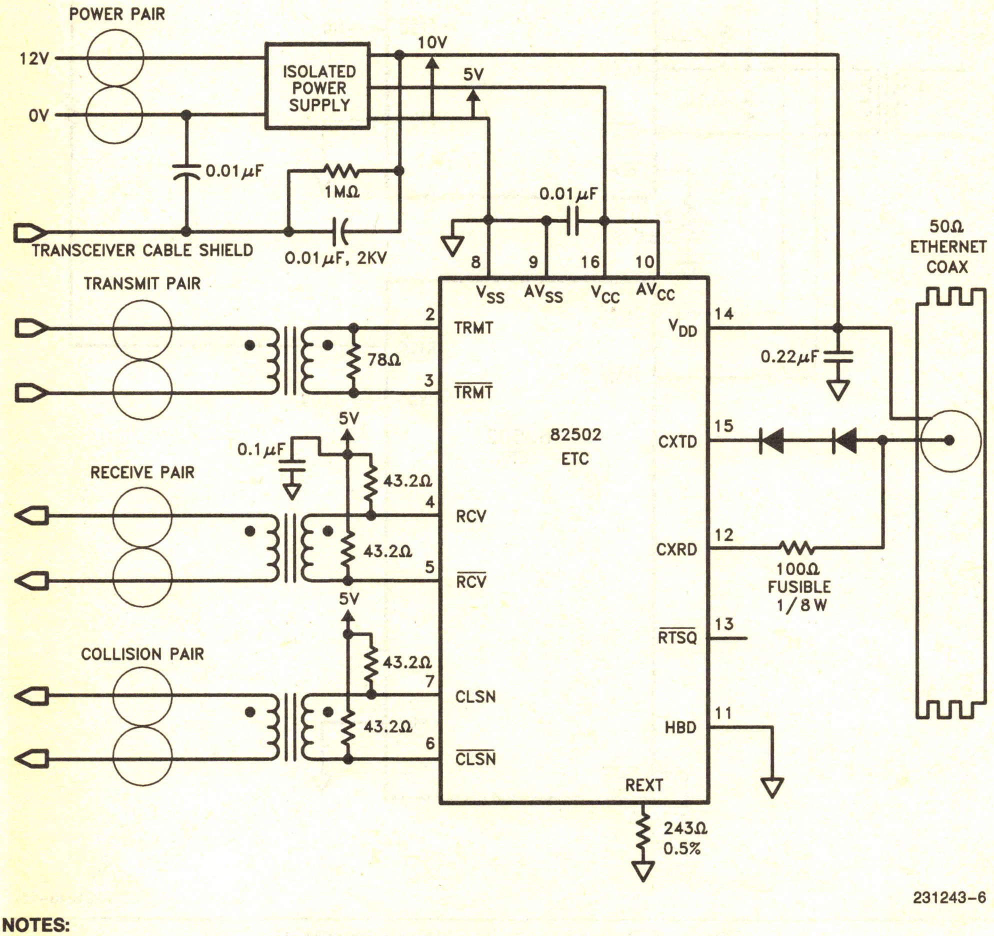
If the HBD input is strapped low, the self test feature of the 82502 is enabled. After each frame transmission, the device transmits a collision presence signal (heartbeat) onto the collision presence pair to indi-

cate that the collision circuitry is operational. The heartbeat begins between 600 ns to 1600 ns following the last positive transition on the transmit pair. The signal's duration is between 0.5 μs and 1.5 μs.

If the HBD input is strapped high, the self test feature as well as the 6.4 μs transmit inhibit timer are disabled.

Design Example

Figure 3 shows the schematic of an IEEE 802.3, 10BASE5 (Ethernet) compatible transceiver based on the 82502. Power to the 82502 is supplied by an isolated DC-to-DC converter which generates +5V ±10% and +10V ±10%. (These voltage levels are referenced to V_{SS}; remember that V_{DD} is connected to earth ground.) Small signal, high conductance diodes which have very small capacitance (less than 3 pF at V_d = 0) are connected between the CXTD input pin and the center conductor to reduce the capacitance load on the channel. A 100Ω fusible resistor on the CXRD pin is to protect the network from being brought down in the event of a short circuit fault between the CXRD pin and GND.



NOTES:
1. Pulse transformer inductance ≥ 50 μH, typically 75 μH
2. Isolated power supply: Pulse Engineering Inc. part no. PE-64369
   Reliability Inc. part nos. 2E12R10-5, 2VA12U10-5, and 2VA5U10-5.
3. Isolated power supply may require additional decoupling capacitors on its inputs and outputs.

Figure 3. Typical 10BASE5 (Ethernet) Transceiver Implementation Using the 82502

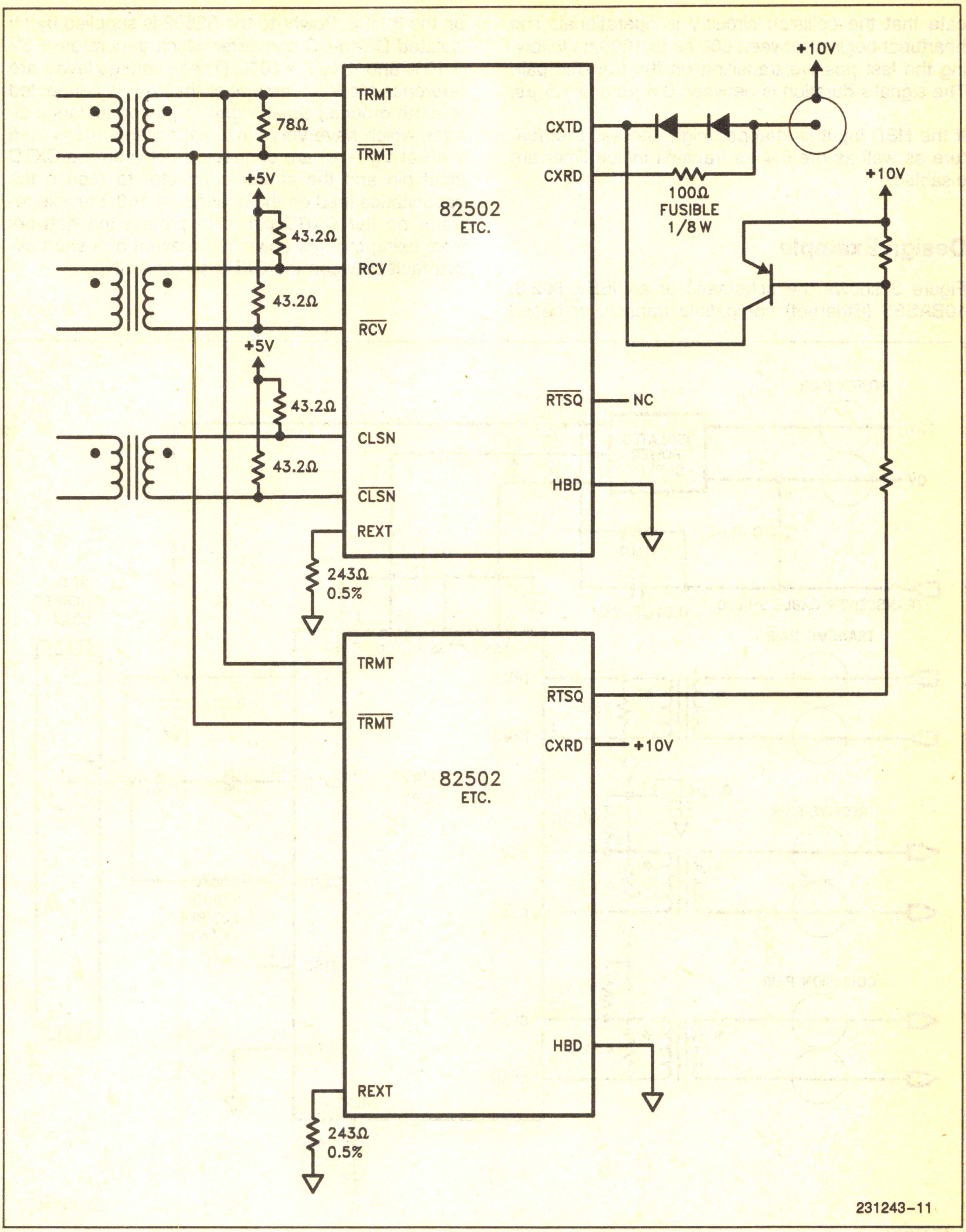


Figure 4. Redundant Transmit Squelch Circuit

Applications

The 82502 is intended for use in high performance, 10 Mbps LAN applications such as IEEE 802.3 10 Base 5 (Ethernet). Ethernet requires that the 82502 transceiver chip be located in a tap box attached directly to the coaxial cable of the network. A drop cable up to 50 meters in length connects the transceiver tap box to the data terminal equipment (DTE), see Figure 5.

In Cheapernet applications, a.k.a. IEEE 802.3 10 Base 2, the 82502 would be located inside the DTE, and transformer coupled to the 82501, see Figure 6. In both applications, the IEEE specifications require that a DC isolated power supply power the 82502.

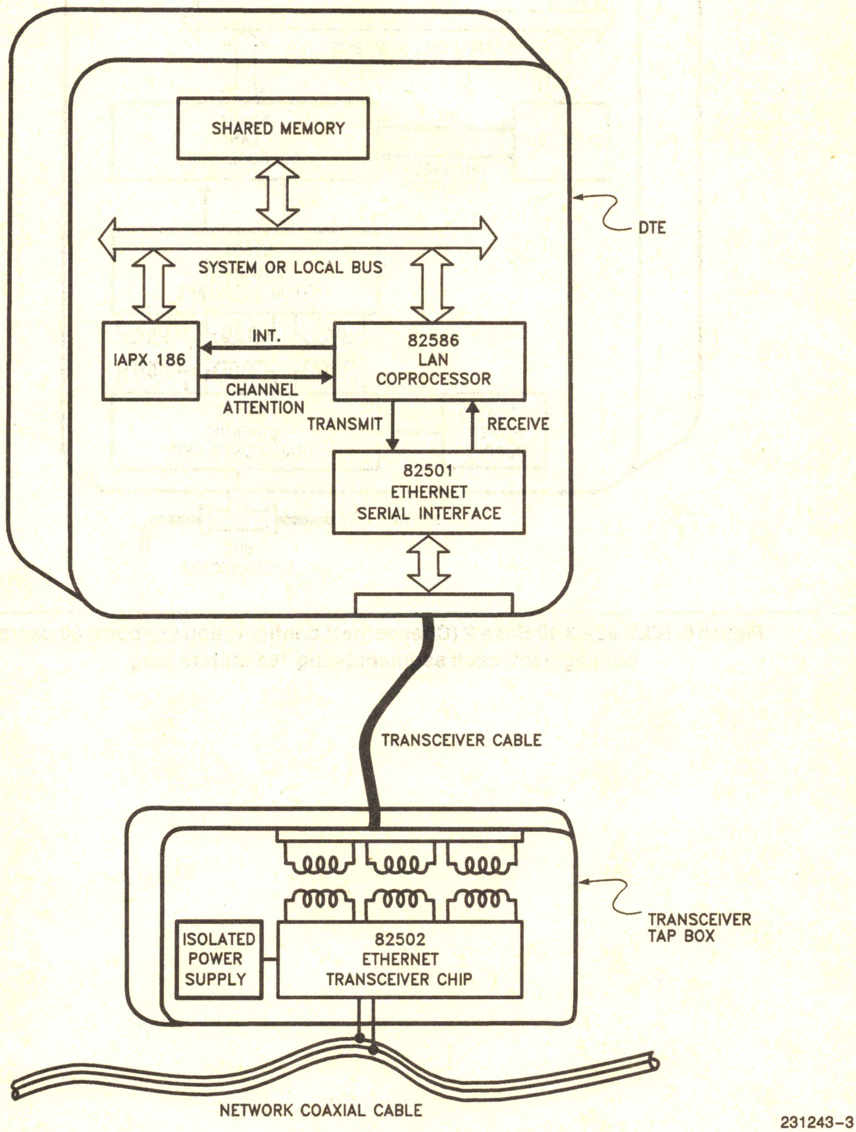


Figure 5. IEEE 802.3 10 Base 5 (Ethernet) configuration supports 100 users per segment, each segment being 500 meters long

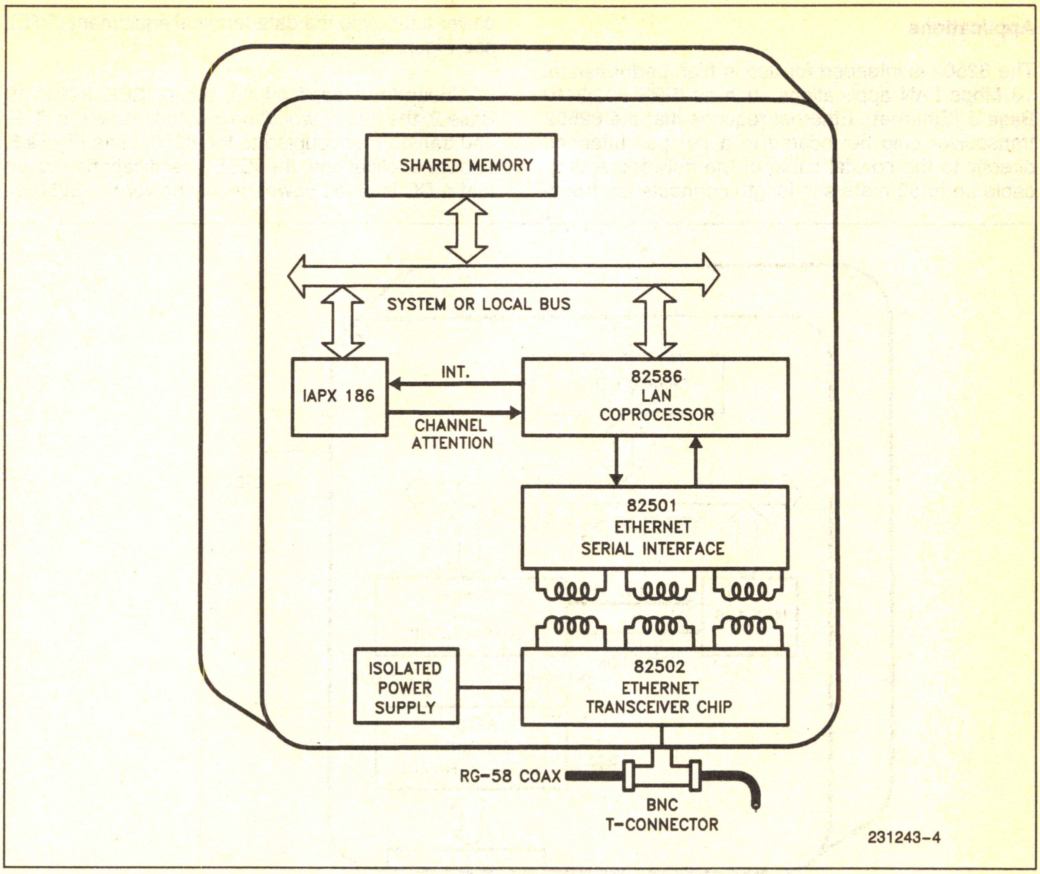


Figure 6. IEEE 802.3 10 Base 2 (Cheapernet) configuration supports 30 users per segment, each segment being 185 meters long.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
V_{DD} with Respect to V_{SS} .... −0.5V to +15V
V_{CC} with Respect to V_{SS} .... −0.5V to +11V
All Input and Output Voltages (Except CXTD and CXRD) with Respect to V_{SS} .... −0.5V to +11V
CXTD Output Voltage with Respect to V_{SS} .... −0.5V to +15V
CXRD Input Voltage with Respect to V_{DD} .... 0V to −15V, > 0V if $I_{CXRD} \leq 200$ mA
Power Dissipation 1.0W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

Operating maximum ratings assume the device transmitting 90% of its operating time.

D.C. CHARACTERISTICS T_A = 0°C–70°C, T_{CMAX} = 115°C, V_{CC} = 5V ±10%, V_{DD} = 10V ±10%

Symbol	Parameter	Limits			Units	Conditions
		Min	Typ	Max		
V_{IDF}	Transceiver Cable Input Differential Voltage	±250		±1200	mV	Base to peak
I_{AVEC}	CXTD Output Current DC Component (Including the effects of timing distortion)	37		45	mA	CXTD: 3.3V to 10.3V
I_{ACC}	CXTD Output Current AC Component	±28	±31	$\pm I_{AVEC}$	mA	
I_{IDC}	CXTD Idle Output Current			5	μA	CXTD: 0V to 11V
I_{OLSQ}	$\overline{RTSQ}$ Output Low Current			15	mA	
V_{OLSQ}	$\overline{RTSQ}$ Output Low Voltage			1	V	I_{OLSQ} = 13 mA
I_{OHSQ}	$\overline{RTSQ}$ Output High Current			100	μA	
V_{ICAC}	Coax Receiver A.C. Voltage	±200			mV	Base to peak
I_{ILC}	Input Leakage Current CXRD	−1		20	μA	
V_{ODF}	Differential Output Swing RCV and CLSN	±0.58		±1.0	V	R_L = 73Ω to 83 Ω ±1% R_{PULLUP} = 43.2Ω ±1% L ≥ 25 μH
I_{IDT}	Output Idle Current Transceiver Cable	3		7 0.2	mA mA	$\overline{RCV}$ and $\overline{CLSN}$ RCV and CLSN
I_{CC}	V_{CC} and AV_{CC} Supply Current			23	mA	
I_{DD}	V_{DD} Supply Current			43	mA	
R_I	Shunt Resistance between V_{DD} and CXRD	1			MΩ	
C_I	Shunt Capacitance between V_{DD} and CXRD		2	3	pF	f = 10 MHz

A.C. CHARACTERISTICS

A.C. Measurement Conditions

I. T_A = 0°C to 70°C, T_{CMAX} = 115°C, V_{CC} = 5V ±10%, V_{DD} = 10V ±10%

II. The A.C. measurements are taken at the following voltage levels for the various kinds of inputs and outputs:

A. Differential Transceiver Cable Inputs and Outputs:

All delay and timing distortion measurements are referenced to the 0V points measured differentially. Rise and fall times are referenced to the 20% and 80% points. The differential voltage swing at the inputs is at least ±275 mV with rise and fall times of 0 to 10 ns.

B. Coaxial Cable Inputs and Outputs:

Delay measurements are referenced to the 50% points of the signals' total swing. V_{DD} is used as the voltage reference. Rise and fall times are measured at the 20% and 80% points. The differential voltage swing at the inputs is at least ±200 mV. The input signal is either a trapezoidal signal with 25 ±5 ns rise/fall times or a sine wave.

In order to simulate a collision, DC voltage of −2.2V referenced to V_{DD} is applied to the CXRD pin.

C. Redundant Transmit Squelch Output:

Delay measurements are referenced to the 50% point of the total swing.

III. The A.C. loading for various outputs is as follows:

A. Transceiver cable drive pins:

RCV, $\overline{\text{RCV}}$, CLSN and $\overline{\text{CLSN}}$ are each loaded with 10 pF to ground and 43.2Ω to V_{CC}. A parallel load of 25 μH minimum and 78Ω is connected between RCV and $\overline{\text{RCV}}$. The same load is present on CLSN and $\overline{\text{CLSN}}$.

B. Coaxial Cable Driver Pins:

A 25Ω resistor is connected between V_{DD} and CXRD. One small signal diode is connected between the CXRD and CXTD pins such that it conducts when the 82502 is transmitting.

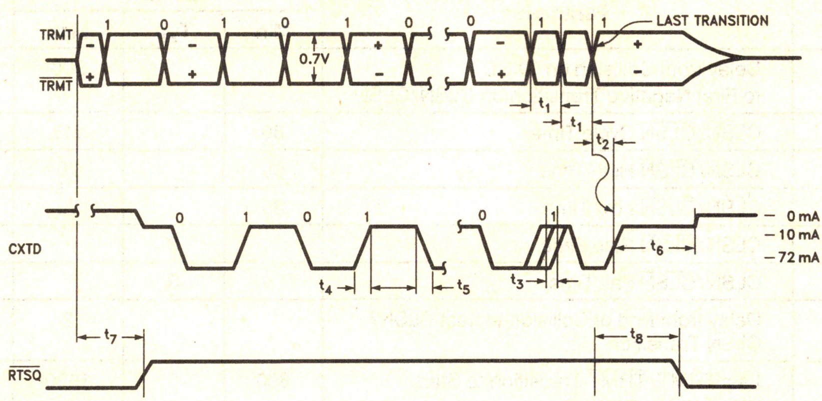
C. Redundant Transfer Squelch Output:

A 680Ω resistor is connected between $\overline{\text{RTSQ}}$ and V_{DD}.

TRANSMIT TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_1	TRMT/$\overline{\text{TRMT}}$ Input Pulse Width	44		106	ns
t_2	Transmit Steady State Delay			50	ns
t_3	CXTD Output Edge Jitter			±2	ns
t_4	CXTD Rise Time	20		30	ns
t_5	CXTD Fall Time	20		30	ns
t_6	Last Data Transition to CXTD Disabled			250	ns
t_7	$\overline{\text{RTSQ}}$ Inactive Delay from First Valid Negative Data Transition on TRMT Pair			200	ns
t_8	$\overline{\text{RTSQ}}$ Active Delay from Last Positive Data Transition on TRMT Pair			325	ns

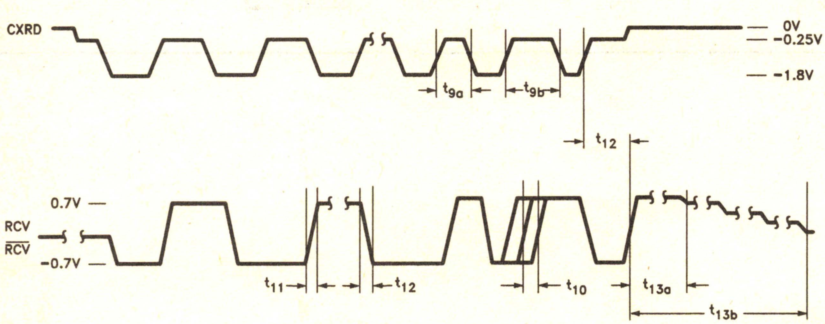
TRANSMIT TIMING (Continued)



231243–7

RECEIVE TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_9	CXRD Input Pulse Width a) Manchester Short Pulse b) Manchester Long Pulse	 28 68		 72 132	 ns ns
t_{10}	RCV/$\overline{RCV}$ Output Edge Jitter			±6	ns
t_{11}	RCV/$\overline{RCV}$ Rise Time		3	6	ns
t_{12}	RCV/$\overline{RCV}$ Fall Time		3	6	ns
t_{13}	Last RCV/$\overline{RCV}$ Transition to Idle a) RCV/$\overline{RCV}$ Held High from Last Postive Transition b) RCV/$\overline{RCV}$ Becomes Idle	 200 		 8000	 ns ns
t_{14}	Receive Steady-State Propagation Delay			50	ns

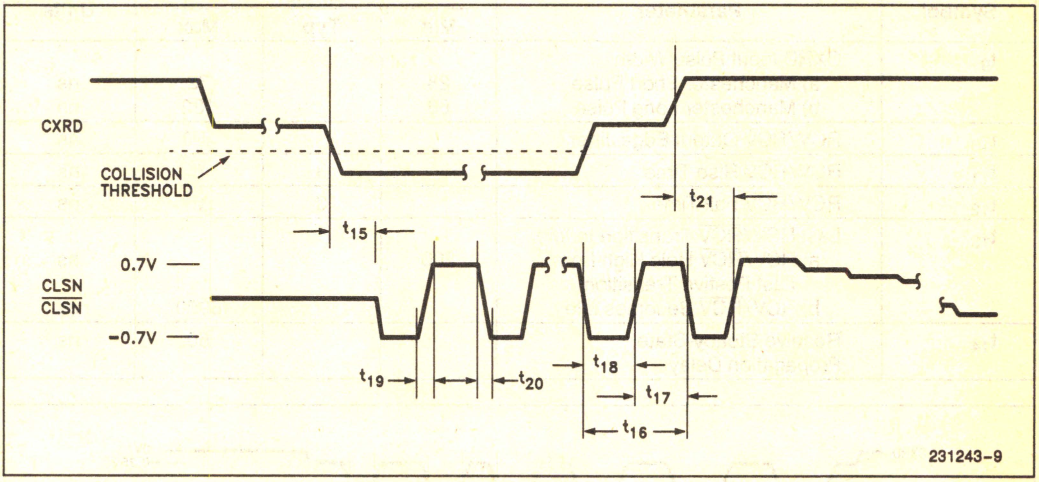


231243–8

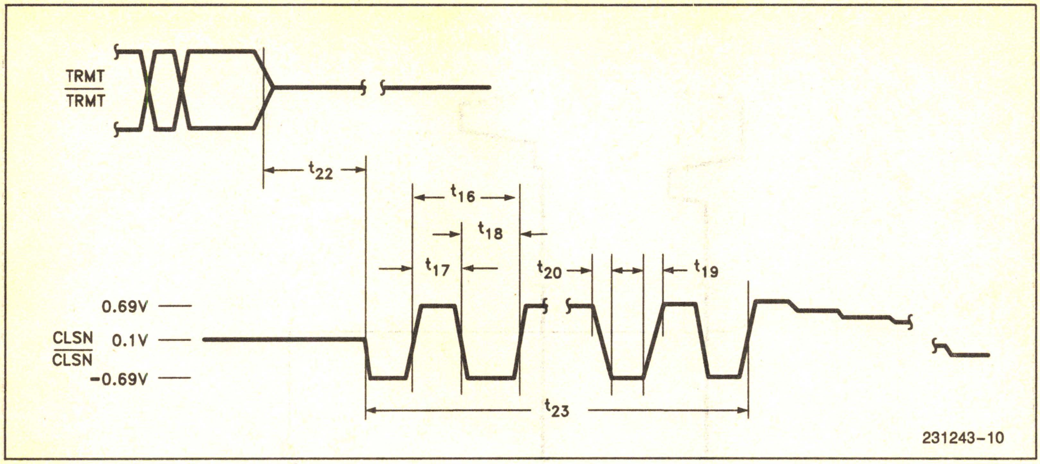
COLLISION AND HEARTBEAT TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_{15}	Delay from Collision on CXRD to First Negative Transition on CLSN/$\overline{\text{CLSN}}$			900	ns
t_{16}	CLSN/$\overline{\text{CLSN}}$ Cycle Time	86		117	ns
t_{17}	CLSN/$\overline{\text{CLSN}}$ High Time	35		70	ns
t_{18}	CLSN/$\overline{\text{CLSN}}$ Low Time	35		70	ns
t_{19}	CLSN/$\overline{\text{CLSN}}$ Rise Time		3		ns
t_{20}	CLSN/$\overline{\text{CLSN}}$ Fall Time		3		ns
t_{21}	Delay from End of Collision to Last CLSN/$\overline{\text{CLSN}}$ Transition			2	μs
t_{22}	Last TRMT/$\overline{\text{TRMT}}$ Transition to Start of Heartbeat Packet	600		1600	ns
t_{23}	Heartbeat Packet Duration	500		1500	ns

COLLISION TIMING



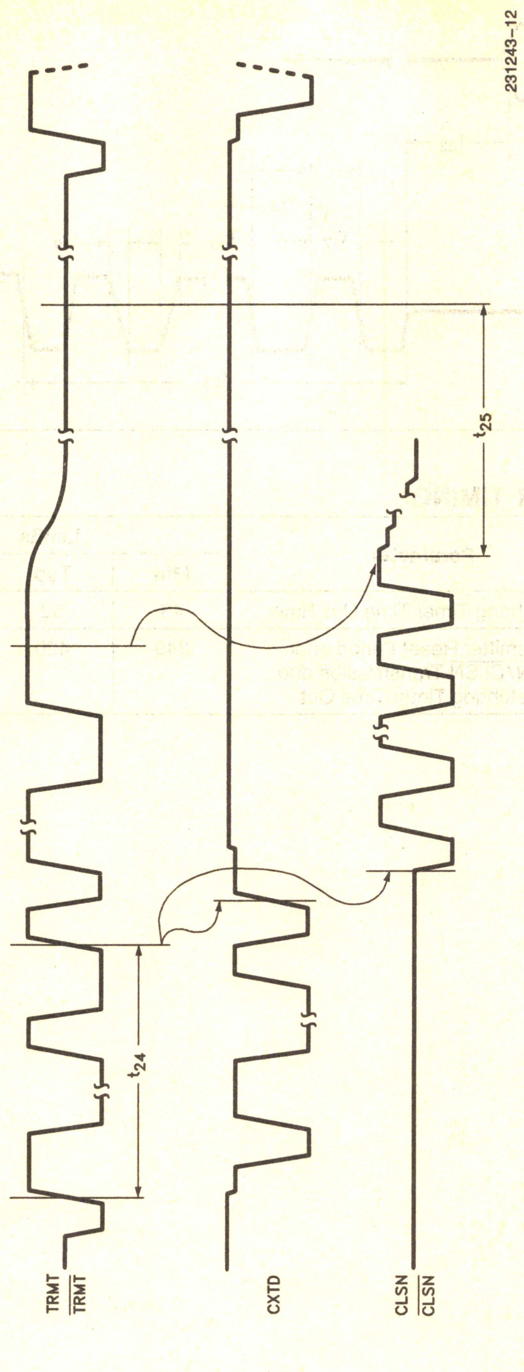
HEARTBEAT TIMING



WATCHDOG TIMER TIMING

Symbol	Parameter	Limits			Units
		Min	Typ	Max	
t_{24}	Watchdog Timer Time Out Time	43	52	61	ms
t_{25}	Transmitter Reset Period after CLSN/$\overline{\text{CLSN}}$ Transmission due to Watchdog Timer Time Out	340	420	550	ms

WATCHDOG TIMER TIMING



82588
High Integration LAN Controller

- **Integrates ISO Layers 1 and 2**
  - **— CSMA/CD Data Link Controller**
  - **— On-Chip Manchester, NRZI Encoding/Decoding**
  - **— On-Chip Logic Based Collision Detect and Carrier Sense**
- **Supports Emerging Industry Standards**
  - **— StarLAN (IEEE 802.3 1BASE5)**
  - **— 2 Mb/s Broadband**
- **High Level Command Interface Offloads the CPU**
- **Efficient Memory Use Via Multiple Buffer Reception**
- **User Configurable**
  - **— Up to 2 Mb/s Bit Rates with On-chip Encoder/Decoder (High Integration Mode)**
  - **— Up to 5 Mb/s with External Encoder/ Decoder (High Speed Mode)**
- **No TTL Glue Required with iAPX 186 and 188 Microprocessors**
- **Network Management and Diagnostics**
  - **— Short or Open Circuit Localization**
  - **— Station Diagnostics (External Loopback)**
  - **— Self Test Diagnostics Internal Loopback User Readable Registers**

The 82588 is a highly integrated device designed for realizing cost sensitive, mid-range Local Area Network (LAN) applications such as personal computer networks.

At data rates of up to 2 Mb/s, it provides a highly integrated interface and performs: CSMA/CD Data Link Control, Manchester, Differential Manchester or NRZI encoding/decoding, clock recovery; Carrier Sense, and Collision Detection. This mode is called "High Integration Mode." In the 82588 "High Speed Mode", the user can transfer data at a rate of up to 5 Mb/s. In this mode the physical link functions are done external to the 82588.

The 82588 is available in a 28 pin DIP and 44 lead PLCC package and fabricated in Intel's reliable HMOS II 5 volt technology.

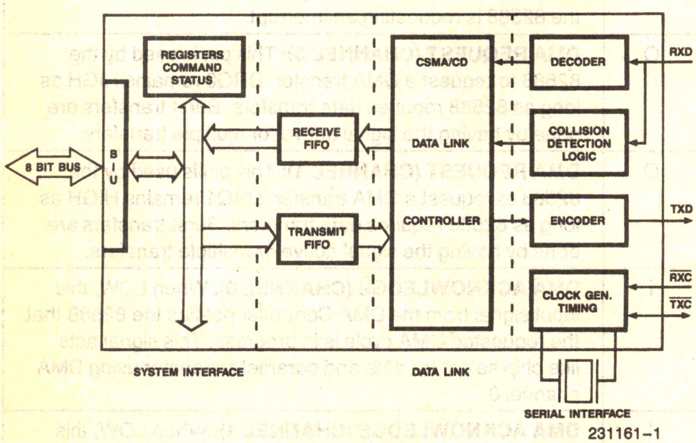
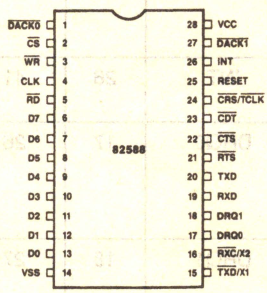
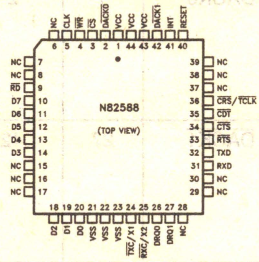


Figure 1. 82588 Block Diagram



231161-2

Figure 2. 82588 Pin Configuration (DIP)



231161-31

Figure 3. 82588 Pin Configuration (PLCC)

November 1986
Order Number: 231161-003

Table 1. Pin Description

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
D7 D6 D5 D4 D3 D2 D1 D0	6 7 8 9 10 11 12 13	10 11 12 13 14 18 19 20	I/O	**DATA BUS:** The Data Bus lines are bi-directional three state lines connected to the system's Data Bus for the transfer of data, commands, status and parameters.
$\overline{RD}$	5	9	I	**READ:** Together with $\overline{CS}$, $\overline{DACK0}$ or $\overline{DACK1}$, Read controls data or status transfers out of the 82588 registers.
$\overline{WR}$	3	4	I	**WRITE:** Together with $\overline{CS}$, $\overline{DACK0}$ or $\overline{DACK1}$, Write controls data or command transfers into the 82588 registers.
$\overline{CS}$	2	3	I	**CHIP SELECT:** When this signal is LOW, the 82588 is selected by the CPU for transfer of command or status. The direction of data flow is determined by the $\overline{RD}$ or $\overline{WR}$ inputs.
CLK	4	5	I	**CLOCK:** System clock. TTL compatible signal.
RESET	25	40	I	**RESET:** A HIGH signal on this pin will cause the 82588 to terminate current activity. This signal is internally synchronized and must be held HIGH for at least four Clock cycles.
INT	26	41	O	**INTERRUPT:** Active HIGH signal indicates to the CPU that the 82588 is requesting an interrupt.
DRQ0	17	26	O	**DMA REQUEST (CHANNEL 0):** This pin is used by the 82588 to request a DMA transfer. DRQ0 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active for multiple transfers.
DRQ1	18	27	O	**DMA REQUEST (CHANNEL 1):** This pin is used by the 82588 to request a DMA transfer. DRQ1 remains HIGH as long as 82588 requires data transfers. Burst transfers are done by having the signal active or multiple transfers.
$\overline{DACK0}$	1	2	I	**DMA ACKNOWLEDGE (CHANNEL 0):** When LOW, this input signal from the DMA Controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 0.
$\overline{DACK1}$	27	42	I	**DMA ACKNOWLEDGE (CHANNEL 1):** When LOW, this input signal from the DMA controller notifies the 82588 that the requested DMA cycle is in progress. This signal acts like chip select for data and parameter transfer using DMA channel 1.

Table 1. Pin Description (Continued)

Symbol	Pin No. DIP	Pin No. PLCC	Type	Name and Function
				High Integration Mode
X1/X2	15/16	24/25	I	**OSCILLATOR INPUTS:** These inputs may be used to connect a quartz crystal that controls the internal clock generator for the serial unit. X1 may also be driven by a MOS level clock whose frequency is 8 or 16 times the bit rate of Transmit/Receive data. X2 must be left floating if X1 has an external MOS clock.
				High Speed Mode
$\overline{TxC}$	15	24	I	**TRANSMIT CLOCK:** This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ encoding, data is transferred to the TxD pin on the HIGH to LOW clock transition. For Manchester encoding the transmitted bit center is aligned with the $\overline{TxC}$ LOW to HIGH transition.
$\overline{RxC}$	16	25	I	**RECEIVE CLOCK:** This signal provides timing information to the internal serial logic. NRZ data should be provided for reception (RxD). The state of the RxD pin is sampled on the HIGH to LOW transition of $\overline{RxC}$. The operating mode of the 82588 is defined when configuring the chip.
$\overline{TCLK}/\overline{CRS}$	24	36	I	In High Speed Mode, this pin is Carrier Sense, input CRS, and is used to notify the 82588 that there is activity on the serial link.
			O	In High Integration Mode, this pin is Transmit Clock, $\overline{TCLK}$, and is used to output the transmit clock.
$\overline{CDT}$	23	35	I	**COLLISION DETECT:** This input notifies the 82588 that a collision has occurred. It is sensed only if the 82588 is configured for external Collision Detect (external circuitry is then required for detecting the collision).
RxD	19	31	I	**RECEIVE DATA:** This pin receives serial data.
TxD	20	32	O	**TRANSMIT DATA:** This pin transmits data to the Serial Link. This signal is HIGH when not transmitting.
$\overline{RTS}$	21	33	O	**REQUEST TO SEND:** When this signal is LOW, the 82588 notifies an external interface that it has data to transmit. It is forced HIGH after a reset and when transmission is stopped.
$\overline{CTS}$	22	34	I	**CLEAR TO SEND:** CTS enables the 82588 to start transmitting data. Raising this signal to HIGH stops the transmission.
VCC	28	1, 43, 44		**POWER:** +5V Supply
VSS	14	21, 22, 23		Ground

Table 1. Pin Description (Continued)

Symbol	Pin No.		Type	Name and Function
	DIP	PLCC		
NC		6 7 8 15 16 17 28 29 30 37 38 39		**NO CONNECT:** These pins are reserved for future use.

FUNCTIONAL DESCRIPTION

High Integration Mode

The 82588 LAN Controller is a highly integrated device designed for cost sensitive LAN applications such as personal workstation cluster networks. Included on the chip is a programmable CSMA/CD controller, an NRZI and Manchester encoder/decoder with clock recovery, and two collision detection mechanisms. With the addition of simple transceiver line drivers or RF Modem, the 82588 takes care of all the major functions of the ISO Physical and Data Link Layers.

CSMA/CD Controller

The 82588 on-chip CSMA/CD controller is programmable, which allows it to operate in a variety of LAN environments including emerging industry standards such as StarLAN (IEEE 802.3 1BASE5) and 2 Mb/s broadband (IBM PC Network). Programmable parameters include:

- Framing (End of Carrier of SDLC)
- Address field length
- Station priority
- Interframe spacing
- Slot time
- CRC-32 OR CRC-16

Encoder/Decoder

The on-chip NRZI and Manchester encoder/decoder supports data rates up to 2 Mb/s. Manchester encoding is often times used in baseband applications and NRZI is used in broadband applications.

Collision Detection

A major innovation with the 82588 is its on-chip logic based collision detection. To ensure high probability of collision detection two mechanisms are provided. The Code Violation method defines a collision when a transition edge occurs outside the area of normal transitions as specified by either the Manchester or NRZI encoding methods. Bit Comparison method compares the signature of the transmitted frame to the receive frame signature (re-calculated by the 82588 while listening to itself). If the signatures are identical the frame is assumed to have been transmitted without a collision.

System Interface

The 82588 goes beyond providing the designer with the functions necessary for interfacing to the LAN. It has an extremely friendly system interface that makes it easy to design with. First, the 82588 has a high level command interface; that is the CPU sends the 82588 commands such as Transmit or Configure. This means the designer does not have to write low level software to perform these tasks, and it offloads the CPU in the application. Second, the 82588 supports an efficient memory structure called Multiple Buffer Reception in which buffers are chained together while receiving frames. This is an important feature in applications with limited memory such as personal computers. Third, the 82588 has two independent sixteen byte FIFO's one for reception and one for transmission. The FIFO's allow the 82588 to tolerate bus latency. Finally the 82588 provides an eight byte data path that supports up to 4 Mbytes/second using external DMA.

Network Management & Diagnostics

The 82588 provides a rich set of diagnostic and network management functions including: internal and external loopback, channel activity indicators, optional capture of all frames regardless of destination address (Promiscuous Mode), capture of collided frames, (if address matches), and time domain reflectometry for locating fault points in the network cable. The 82588 Register Dump Command ensures reliable software by dumping the content of the 82588 registers into the system memory.

The next section will describe the 82588 system bus interface, the 82588 network interface, and the 82588 internal architecture.

82588/Host CPU Interaction

The CPU communicates with the 82588 through the system's memory and 82588's on-chip registers. The CPU creates a data structure in the memory, programs the external DMA controller with the start address and byte count of the block, and issues the command to the 82588.

The 82588 is optimized for operating with the iAPX 186/188, but due to the small number of hardware signals between the 82588 and the CPU, the 82588 can operate easily with other processors. The data bus is 8 bits wide and there is no address bus.

Chip Select and Interrupt lines are used to communicate between the 82588 and the host as shown in the Figure 3. Interrupt is used by the 82588 to draw the CPU's attention. The Chip Select is used by the CPU to draw the 82588's attention.

There are two kinds of transfer over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data transfers are requested by the 82588, and are typically performed by a DMA controller. The table given in Figure 4 shows the Command/Status and data transfer control signals.

The CPU writes to 82588 using $\overline{CS}$ and $\overline{WR}$ signals. The CPU reads the 82588 status register using $\overline{CS}$ and $\overline{RD}$ signals.

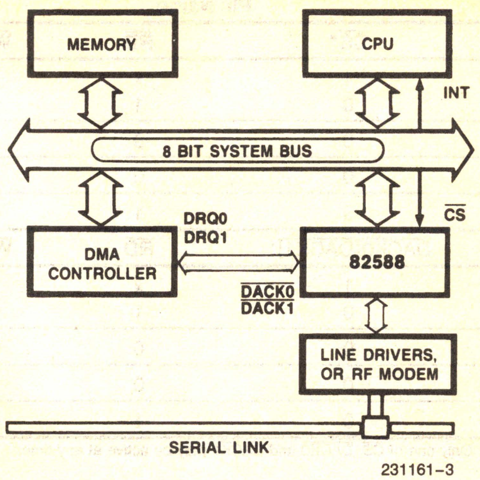


Figure 3. 82588/HOST CPU Interaction

To initiate an operation like Transmit or Configure (see Figure 5), a Write operation command from CPU to 82588 is issued by the CPU. A Read operation from CPU gives the status of the 82588. Although there are four status registers they're read using the same port in a round robin fashion (Figure 6).

Any parameters or data associated with a command are transferred between the memory and 82588 using DMA. The 82588 has two data channels, each having Request and Acknowledge lines. Typically one channel is used to receive data and other to transmit data and perform all the other initialization and maintenance operations like Configure, Address Set-Up, Diagnose, etc. The channels are identical and can be used interchangeably.

When the 82588 requires access to the memory for parameter or data transfer it activates the DMA request lines and uses the DMA controller to achieve the data transfer. Upon the completion of an operation, the 82588 interrupts the CPU. The CPU then reads results of the operation (the status of the 82588).

Pin Name			Function
$\overline{CS}$*	$\overline{RD}$	$\overline{WR}$	
1	x	x	No transfer to/from Command/Status
0	1	1	
0	0	0	Illegal
0	0	1	Read from status register
0	1	0	Write to Command register
$\overline{DACK0}$[$\overline{DACK1}$]*	$\overline{RD}$	$\overline{WR}$	
1	x	x	No DMA transfer
0	1	1	
0	0	0	Illegal
0	0	1	Data Read from DMA channel 0 [or 1]
0	1	0	Data Write to DMA channel 0 [or 1]

* Only one of $\overline{CS}$, $\overline{DACK0}$ and $\overline{DACK1}$ may be active at any time.

Figure 4. Databus Control Signals and Their Functions

7	6 5	4	3 2 1 0
INT. ACK.	PNTR	CHNL	COMMANDS

COMMAND REGISTER

COMMANDS		VALUE
NOP	—	0
IA-SETUP	—	1
CONFIGURE	—	2
MC-SETUP	—	3
TRANSMIT	—	4
TDR	—	5
DUMP	—	6
DIAGNOSE	—	7
RETRANSMIT	—	12

COMMANDS		VALUE
ABORT	—	13
RECEIVER-ENABLE	—	8
ASSIGN NEXT BUF	—	9
RECEIVE-DISABLE	—	10
STOP-RECEPTION	—	11
RESET	—	14
FIX PTR	—	15 (CHNL = 1)
RLS PTR	—	15 (CHNL = 0)

Figure 5. Command Format and Operation Values

	7	6	5	4	3	2	1	0
Status 0	INT.	RCV	EXEC	CHNL	EVENT			
Status 1				RESULT 1				
Status 2				RESULT 2				
Status 3	RCV CHNL	RCV STATE		BUFF CHNG NO. OF BUF		EXEC CHNL	EXEC STATE	

EVENTS		VALUE (STATUS 0)
IA-SETUP-DONE	—	1
CONFIGURE-DONE	—	2
MC-SETUP-DONE	—	3
TRANSMIT-DONE	—	4
TDR-DONE	—	5
DUMP-DONE	—	6
DIAGNOSE-PASSED	—	7
END OF FRAME	—	8
REQUEST NEXT BUFFER	—	9
RECEPTION ABORTED	—	10
RETRANSMIT-DONE	—	12
EXECUTION-ABORTED	—	13
DIAGNOSE-FAILED	—	15

Figure 6. Status Registers and Event Values

Transmitting a Frame

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 7. Its first two bytes specify the length of the rest of the block. The next few bytes (Up to 6 bytes long) contain the destination address of the node it is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 82588.

Upon receiving the command, the 82588 fetches the first two bytes of the block to determine the length of the block. If the link is free, and the first data byte was fetched, the 82588 begins transmitting the preamble and concurrently fetches the bytes from the Transmit Data Block and loads them into a 16 byte FIFO to keep them ready for transmitting. The FIFO is a buffer between the serial and parallel part of the 82588. The on-chip FIFOs help the 82588 to tolerate system bus latency as well as provide efficient usage of system bandwidth.

The destination address is sent out after the preamble. This is followed by the source or the station individual address, which is stored earlier on the 82588 using the IA-SETUP command. After that, the entire information field is transmitted followed by a CRC field calculated by the 82588. If during the transmission of the frame, a collision is encountered, then the transmission is aborted and a jam pattern is sent out after completion of the preamble. The 82588 generates an Interrupt indicating the experience of a collision and the frame has to be re-transmitted. Retransmission is done by the CPU exactly as the Transmit command except the Re-Transmit command keeps track of the number of collisions encountered. When the 82588 gets the Retransmit command and the exponential back-off time is expired, the 82588 transmits the frame again. The transmitted frame can be coded to either Manchester, Differential Manchester or NRZI methods.

Collision Detection

The 82588 eliminates the need for external collision detection logic, in most applications, while easing or eliminating the need for complex transceivers. Two algorithms are used for collision detection: Bit Comparison and Code Violation. The Bit Comparison Method is useful in Broadband networks where there are separate transmit and receive channels. Bit Comparison compares the "signature" of the transmitted data and received data at the end of the collision window in any network configuration. This algorithm calculates the CRC after a programmable number of transmitted bits, holds this CRC in a register, and compares it with received data's CRC. A CRC or "signature" difference indicates a collision. The code violation is detected if the encoding of the received data has any bit that does not fit the encoding rules. The code violation method is useful in short bus topology and serial backplane applications where bit attenuation over the bus is negligible.

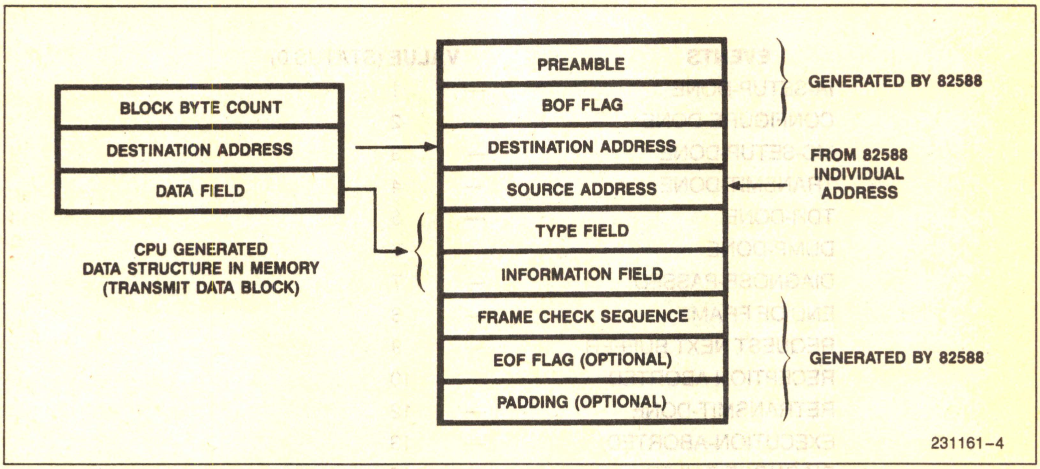


Figure 7. The 82588 Frame Structure and location of Data element in System Memory

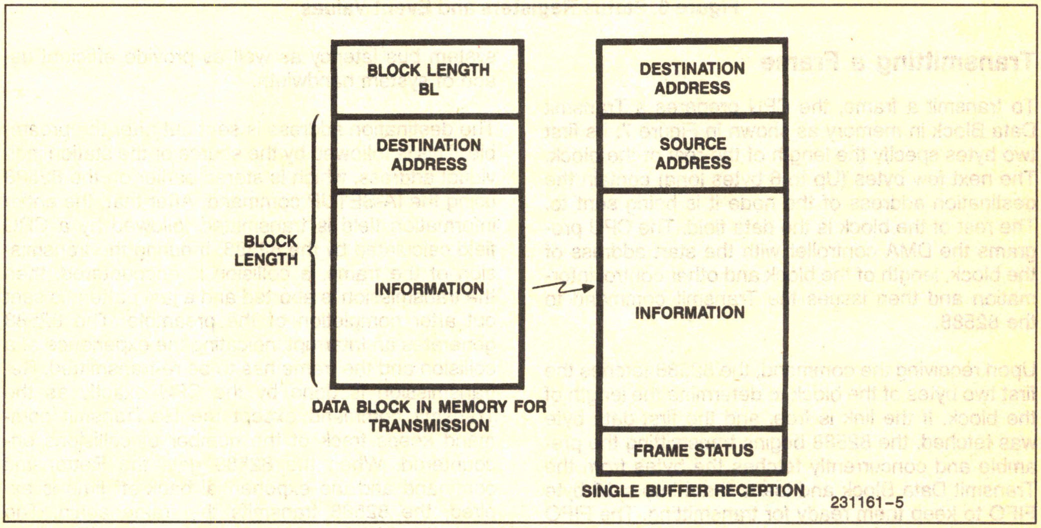


Figure 8. Single Buffer Reception

Receiving a Frame

The 82588 can receive a frame when its receiver has been enabled. The received frame is decoded by either on-chip Manchester, Differential Manchester or NRZI decoders in High Integration Mode and NRZI in High Speed Mode. The 82588 checks for an address match for an individual address, a Mulitcast address or a Broadcast address. In the Promiscuous mode the 82588 receives all frames. Only when the address match is successful does the 82588 transfer the frame to the memory using the DMA controller. Before enabling the receiver, the CPU makes a memory buffer area available to the Receive Unit and programs the starting address of the DMA controller. The received frame is transferred to the memory buffer in the format shown in Figure 8. This method of reception is called "Single Buffer" reception. The entire frame is contained in one continuous buffer. Upon completion of reception the total number of bytes written into the memory buffer is loaded into status registers 1 and 2 and the status of the reception itself is appended to the received frame. An interrupt to the CPU follows.

If the frame size is unknown, memory usage can be optimized by using "Multiple Buffer" reception.

This way the user does not have to allocate large memory space for short frames. Instead, the 82588 can dynamically allocate memory space as it receives frames. This method requires both DMA channels alternately to receive the frame. As the frame reception starts, the 82588 interrupts the CPU and automatically requests assignment of the next sequential buffer. The CPU does this and loads the second DMA channel with the next buffer information so that the 82588 can immediately switch to the other channel as soon as the current buffer is full. When the 82588 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate another buffer on the other (previous) channel in advance. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU using a buffer chain descriptor structure in memory (see Figure 9).

This dynamic (pre) allocation of memory buffers results in efficient use of available storage when handling frames of widely differing sizes. Since the buffers are pre-allocated one block in advance, the system is not time critical.

80188 Based System

Figure 10 shows a high performance, high-integration configuration of the 82588 with the 80188 in a typical iAPX188-based microcomputer. The 80188 controls the 82588, as well as providing DMA control services for data transfer, using its "on chip" two channel DMA controller.

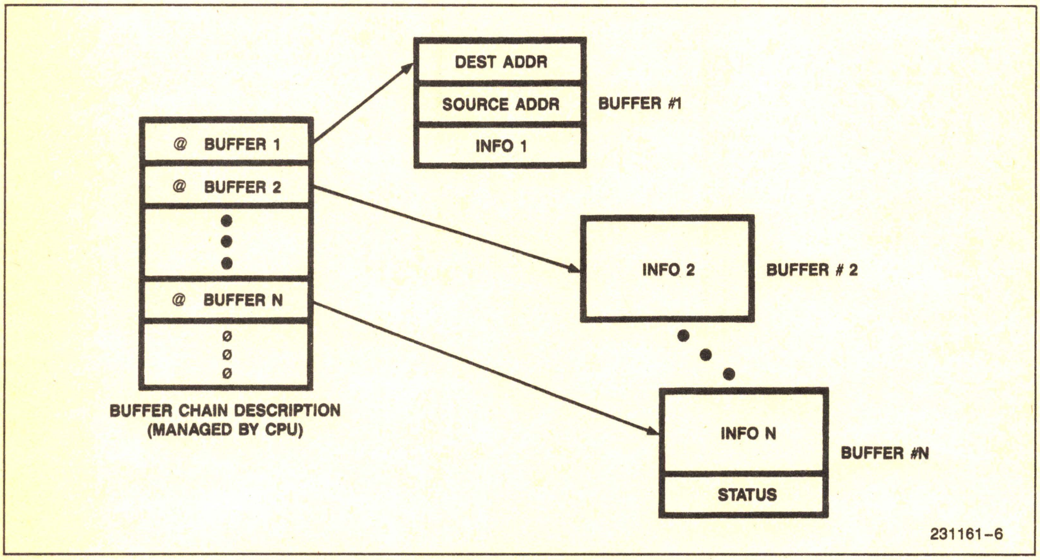


Figure 9. Multiple Buffer Reception

Link Interface

The Serial Interface Mode configuration parameter selects either a highly integrated Direct Link interface (High Integration Mode) or a highly flexible Transceiver Interface (High Speed Mode).

Application

In the High Integration Mode it is possible to connect the 82588 on a very short "Wired OR" link, on a longer twisted pair cable, or a broadband connection.

Twisted Pair Connection

The link consists of a twisted pair that interconnects the 82588. The transmit data pin is connected via a driver and the receive data pin is connected via a buffer. The twisted pair must be properly terminated to prevent reflections.

In the minimum configuration, TxD and RxD are connected to the twisted pair and $\overline{CTS}$ is grounded. The 82588 may control the driver with the $\overline{RTS}$ pin. It is also possible to use external circuitry for performing collision detection, and feeding it to the 82588 through the $\overline{CDT}$ pin.

Broadband Connection

The 82588 supports data communications over a broadband link in both its modes. Proper MODEM interface should be provided. Collision Detection by Bit Comparison, in High Interface Mode, can be applied to transmission over broadband links.

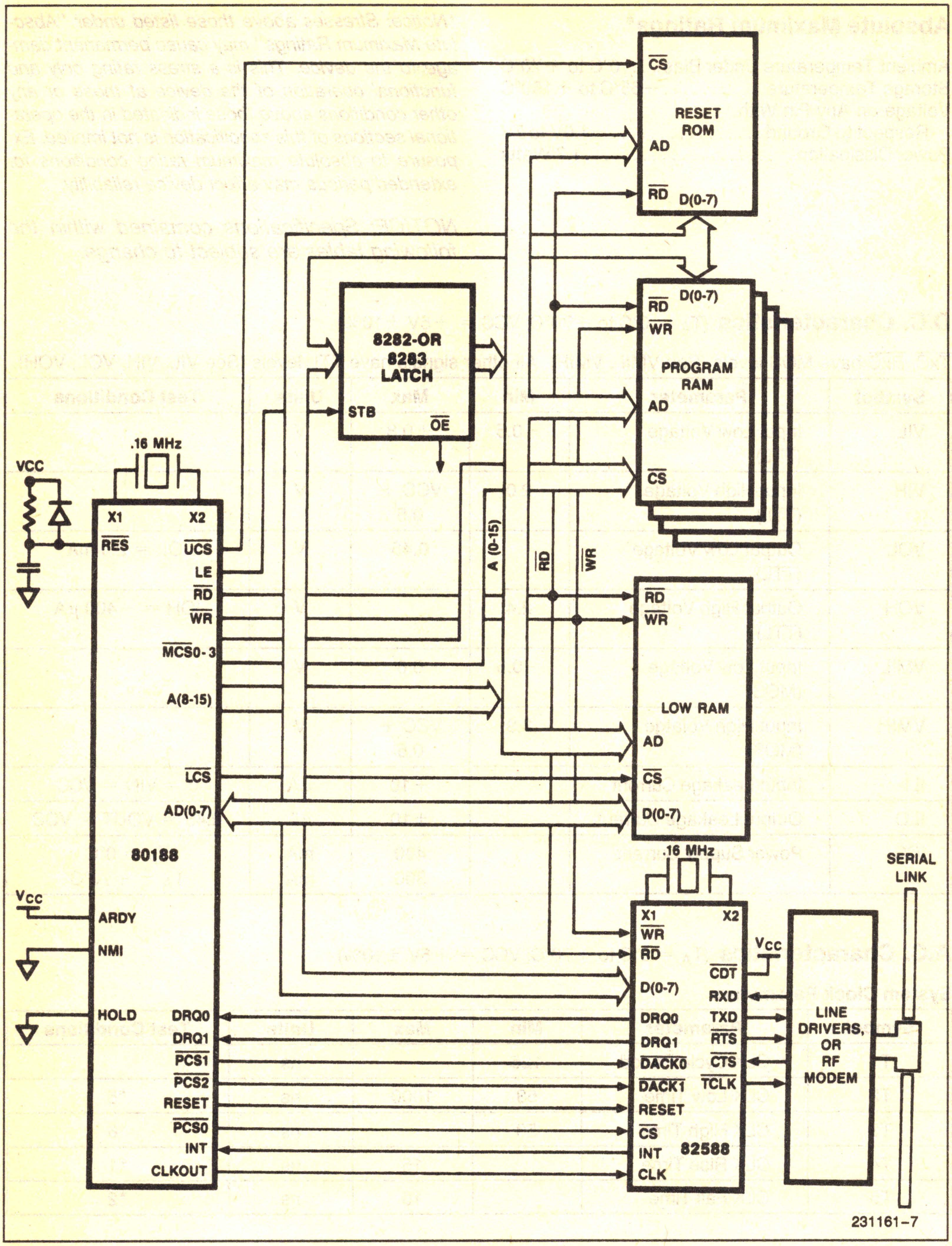


Figure 10. 80188 Based System

Absolute Maximum Ratings*

Ambient Temperature Under Bias . . . . 0°C to +70°C
Storage Temperature . . . . −65°C to +150°C
Voltage on Any Pin With
Respect to Ground . . . . −1.0V to 7V
Power Dissipation . . . . 1.7 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. Characteristics (T_A = 0°C to +70°C; VCC = +5V ±10%)

$\overline{TxC}$, $\overline{RxC}$ have MOS levels (See VMIL, VMIH). All other signals have TTL levels (See VIL, VIH, VOL, VOH).

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage (TTL)	−0.5	+0.8	V	
VIH	Input High Voltage (TTL)	2.0	VCC + 0.5	V	
VOL	Output Low Voltage (TTL)		0.45	V	IOL = 2.0 mA
VOH	Output High Voltage (TTL)	2.4		V	IOH = −400 μA
VMIL	Input Low Voltage (MOS)	−0.5	0.6	V	
VMIH	Input High Volatge (MOS)	3.9	VCC + 0.5	V	
ILI	Input Leakage Current		+10	μA	0 = VIN = VCC
ILO	Output Leakage Current		±10	μA	0.45 = VOUT = VCC
ICC	Power Supply Current		400 300	mA mA	T_A = 0°C T_A = +70°C

A.C. Characteristics (T_A = 0°C to +70°C; VCC = +5V ±10%)

System Clock Parameters

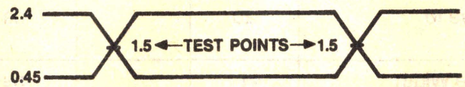
Symbol	Parameter	Min	Max	Units	Test Conditions
T1	CLK Cycle Period	125		ns	
T2	CLK Low Time	53	1000	ns	*5
T3	CLK High Time	53		ns	*6
T4	CLK Rise Time		15	ns	*1
T5	CLK Fall Time		15	ns	*2

A.C. Characteristics (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Reset Parameters					
T6	Reset Active to Clock Low	20		ns	*3
T8	Reset Pulse Width	4T1		ns	
T9	Control Inactve After Reset		T1	ns	
Interrupt Timing Parameters					
T10	CLK High to Interrupt Active		85	ns	*4
T11	$\overline{WR}$ Idle to Interrupt Idle		85	ns	*4
Write Parameters					
T12	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Setup to $\overline{WR}$ Low	0		ns	
T13	$\overline{WR}$ Pulse Width	95		ns	
T14	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Hold After $\overline{WR}$ High	0		ns	
T15	Data Setup to $\overline{WR}$ High	75		ns	
T16	Data Hold After $\overline{WR}$ High	0		ns	
Read Parameters					
T17	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Setup to $\overline{RD}$ Low	0		ns	
T18	$\overline{RD}$ Pulse Width	95		ns	
T19	$\overline{CS}$ or $\overline{DACK0}$ or $\overline{DACK1}$ Address Valid After $\overline{RD}$ High	0		ns	
T20	$\overline{RD}$ Low to Data Valid		80	ns	*7
T21	Data Float After $\overline{RD}$ High		55	ns	*7
DMA Parameters					
T22	CLK Low to DRQ0 or DRQ1 Active		85	ns	*4
T23	$\overline{WR}$ or $\overline{RD}$ Low to DRQ0 or DRQ1 Inactive		60	ns	*4

NOTES:
*1—0.8V–2.0V
*2—2.0V–0.8V
*3—to guarantee recognition at next clock
*4—CL = 50 pF
*5—measured at 1.5V
*6—measured at 1.5V
*7—CL = 20 pF–200 pF

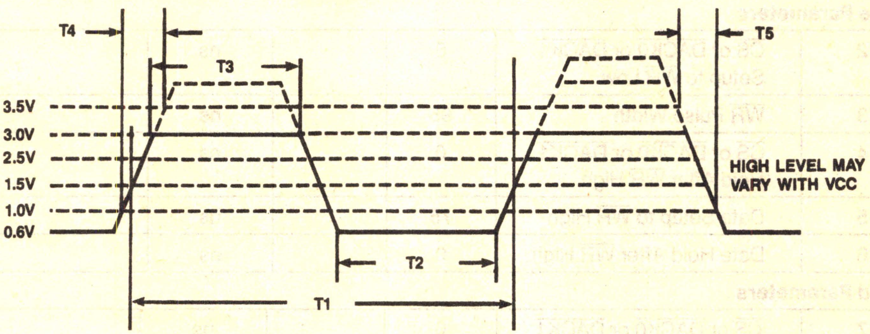
A.C. TESTING INPUT/OUTPUT WAVEFORM



231161-8

AC Testing Inputs are Driven at 2.4V for a Logic 1 and 0.45V for a Logic Q Timing Measurements are Made at 1.5V for Both a Logic 1 and Q:
Rise and Fall Time of Input/Output Signals are Measured Between 0.8V to 2.0V Respectively.

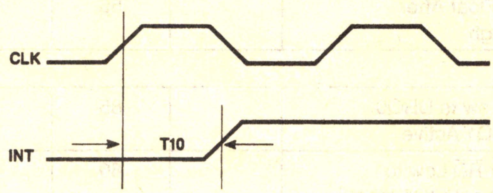
TTL Input/Output Voltage Levels for Timing Measurements



231161-9

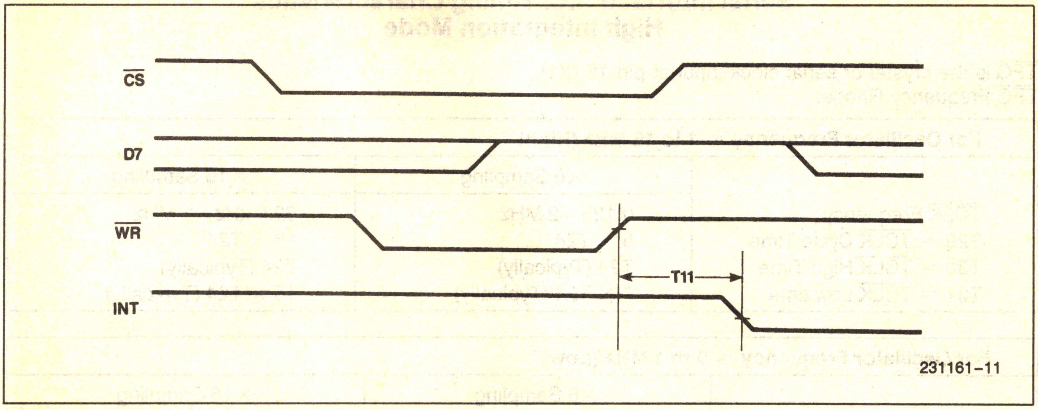
Rise and Fall Time of Input Signals are Measured Between 1.0V to 3.5V Respectively.

Clocks MOS Input Voltage Levels for Timing Measurements

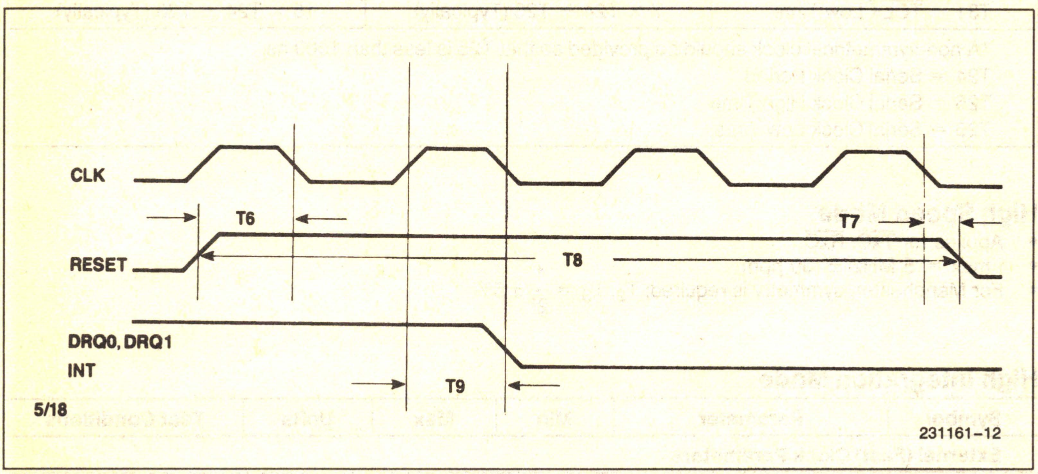


231161-10

Interrupt Timing (Going Active)



Interrupt Timing (Going Inactive)



Reset Timing

Serial Interface A.C. Timing Characteristics
High Integration Mode

$\overline{TFC}$ is the crystal or serial clock input at pin 15 (X1).
$\overline{TFC}$ Frequency Range:

For Oscillator Frequency = 1 to 16 MHz (High)		
	×8 Sampling	×16 Sampling
$\overline{TCLK}$ Frequency	0.125 – 2 MHz	62.5 kHz – 1 MHz
T29 = $\overline{TCLK}$ Cycle Time	8 × T24	16 × T24
T30 = $\overline{TCLK}$ High Time	T24 (Typically)	T24 (Typically)
T31 = $\overline{TCLK}$ Low time	7 × T24 (Typically)	15 × T24 (Typically)

For Oscillator Frequency = 0 to 1 MHz (Low)*		
	×8 Sampling	×16 Sampling
$\overline{TCLK}$ Frequency	0 – 0.125 MHz	0 – 6.25 kHz
T29 = $\overline{TCLK}$ Cycle Time	8 × T24	16 × T24
T30 = $\overline{TCLK}$ High Time	T25 (Typically)	T25 (Typically)
T31 = $\overline{TCLK}$ Low Time	7 × T24 + T26 (Typically)	15 × T24 + T26 (Typically)

*A non-symmetrical clock should be provided so that T25 is less than 1000 ns.
T24 = Serial Clock Period
T25 = Serial Clock High Time
T26 = Serial Clock Low Time

High Speed Mode

- Applies for $\overline{TxC}$, $\overline{RxC}$
- f max = 5 MHz ±100 ppm
- For Manchester, symmetry is required: $T_2, T_3 = \frac{1}{2f} \pm 5\%$

High Integration Mode

Symbol	Parameter	Min	Max	Units	Test Conditions
External (Fast) Clock Parameters					
T24	Fast Clock Cycle	62.5		ns	*1
T25	$\overline{TFC}$ High Time	20	1000	ns	*1, *14
T26	$\overline{TFC}$ Low Time	20		ns	*1
T27	$\overline{TFC}$ Rise Time		5	ns	*1
T28	$\overline{TFC}$ Fall Time		5	ns	*1
Transmit Clock Parameters					
T29	Transmit Clock Cycle	500		ns	*3, *12
T30	$\overline{TCLK}$ High Time	*8	1030	ns	*3
T31	$\overline{TCLK}$ Low Time	*9		ns	*3
T32	$\overline{TCLK}$ Rise Time		15	ns	*3
T33	$\overline{TCLK}$ Fall Time		15	ns	*3

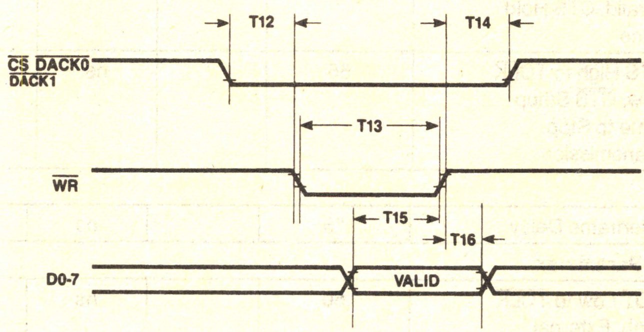
High Integration Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Transmit Data Parameters (Manchester)					
T34	TxD Transition-Transition	4T24-10		ns	*12
T35	$\overline{TCLK}$ Low to TxD Transition Half Bit Cell		*10	ns	*2, *12
T36	$\overline{TCLK}$ Low to TxD Transition Full Bit Cell		*11	ns	*2, *12
T37	TxD Rise Time		15	ns	*2
T38	TxD Fall Time		15	ns	*2
Transmit Data Parameters (NRZI)					
T39	TxD Transition-Transition	8T24-10		ns	*12
T40	$\overline{TCLK}$ Low to TxD Transition		*10	ns	*2, *12
T41	$\overline{TxD}$ Rise Time		15	ns	*2
T42	TxD Fall Time		15	ns	*2
$\overline{RTS}$, $\overline{CTS}$, Parameters					
T43	$\overline{TCLK}$ Low To $\overline{RTS}$ Low		*10	ns	*3, *12
T44	$\overline{CTS}$ Low to $\overline{TCLK}$ Low $\overline{CTS}$ Setup Time	65		ns	
T45	$\overline{TCLK}$ low to $\overline{RTS}$ High		*10	ns	*3, *12
T46	$\overline{TCLK}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	20		ns	*4, *13
T47	$\overline{CTS}$ High to $\overline{TCLK}$ Low. $\overline{CTS}$ Setup Time to Stop Transmission	65		ns	*4
IFS Parameters					
T48	Interframe Delay	*5		ns	
Collision Detect Parameter					
T49	$\overline{CDT}$ Low to $\overline{TCLK}$ High. External Collision Detect Setup Time	50		ns	*13
T50	$\overline{CDT}$ High to $\overline{TCLK}$ Low	50		ns	*13
T51	$\overline{TCLK}$ High to $\overline{CDT}$ Inactive. $\overline{CDT}$ Hold Time	20		ns	*13

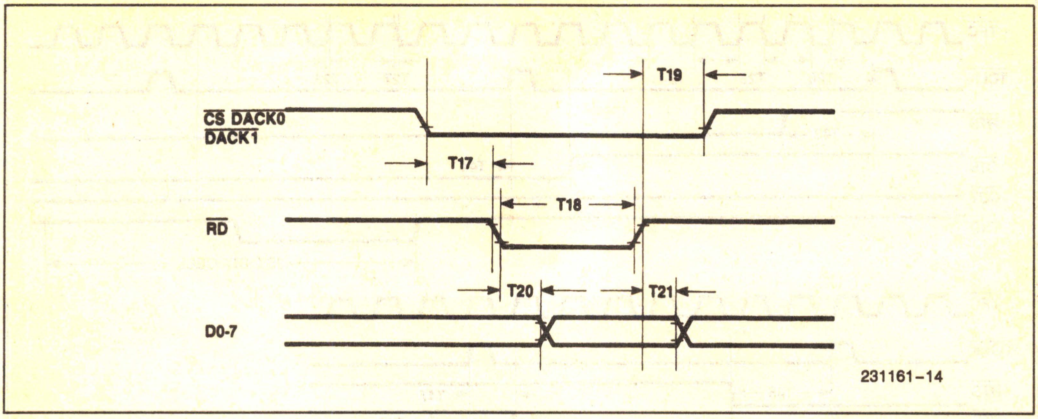
High Integration Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
Collision Detect Parameters (Continued)					
T52	$\overline{CDT}$ Low to Jamming Start		*6	ns	
T53	Jamming Period	*7		ns	
Received Data Parameters (Manchester)					
T54	RxD Transition-Transition	4T24		ns	*12
Received Data Parameters (Manchester)					
T55	RxD Rise Time		10	ns	*1
T56	RxD Fall Time		10	ns	*1
Received Data Parameters (NRZI)					
T57	RxD Transition-Transition	8T24		ns	*12
T58	RxD Rise Time		10	ns	*1
T59	RxD Fall Time		10	ns	*1

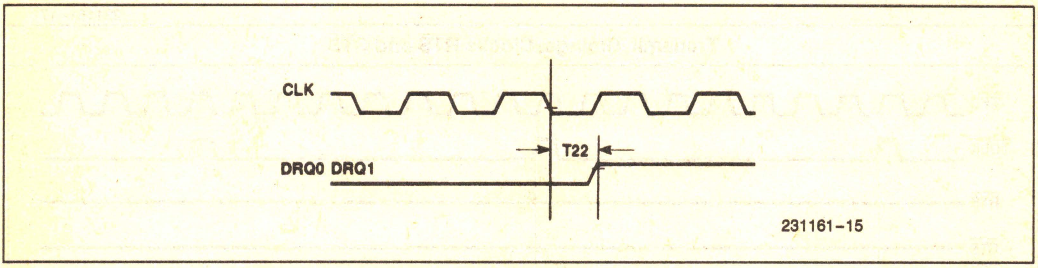
NOTES:
*1—MOS levels.
*2—1 TTL load + 50 pF.
*3—1 TTL load + 100 pF.
*4—Abnormal end to transmission: $\overline{CTS}$ expires before $\overline{RTS}$.
*5—Programmable value: T48 = NIFS × T29 (ns) NIFS—the IFS configuration value.
If NIFS is less than 12, then it is enforced to 12.
*6—Programmable value:
T52 = NCDF × T29 + (12 to 15) × T29 (if collision occurs after preamble).
*7—T53 = 32 × T29
*8—Depends on T24 frequency range:
High Range: T24 − 10
Low Range: T25 − 10
*9—T31 = T29 − T30 − T32 −T33
*10—2T24 + 40 ns
*11—6T24 + 40 ns
*12—For ×16 sampling clock parameter minimum value should be multiplied by a factor of 2.
*13—To guarantee recognition on the next clock.
*14—62.5 ns minimum in Low Range.



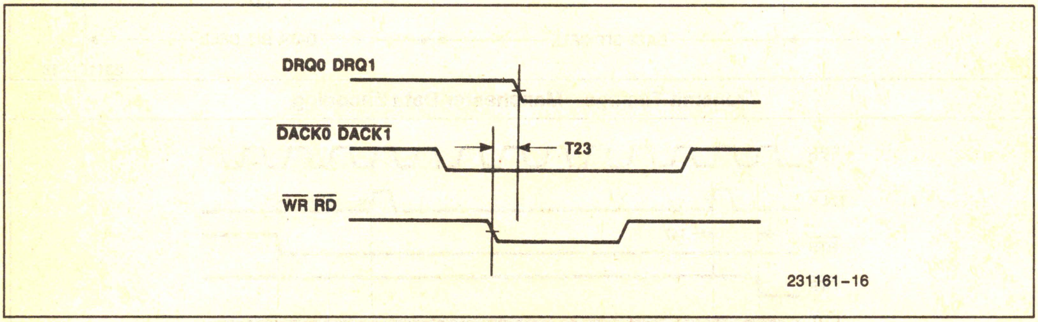
Write Timing



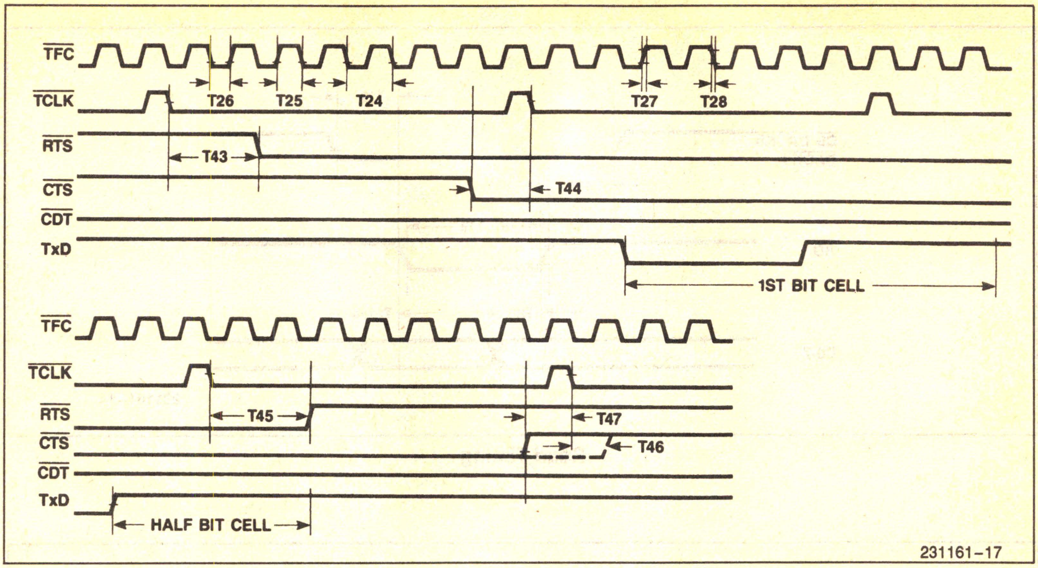
Read Timing



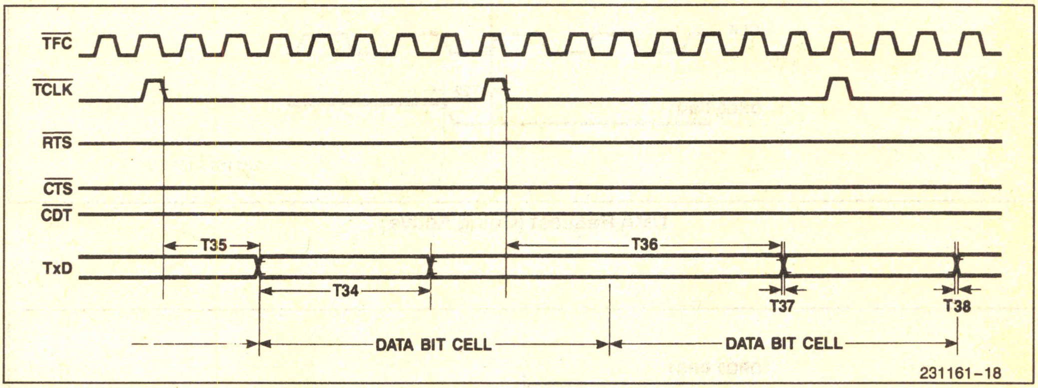
DMA Request (Going Active)



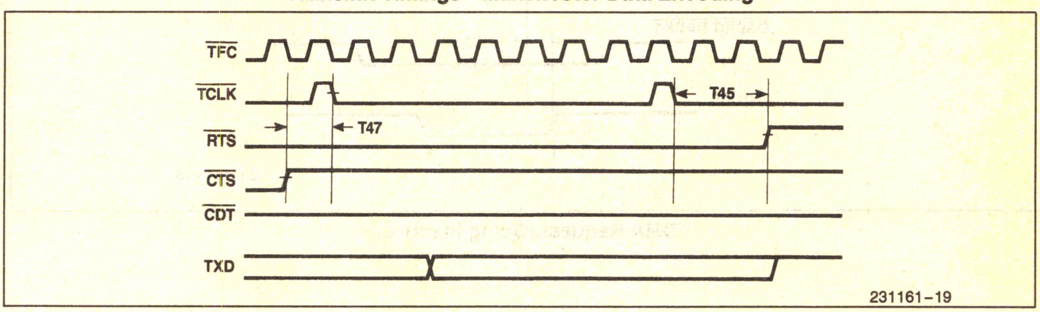
DMA Request (Going Inactive)



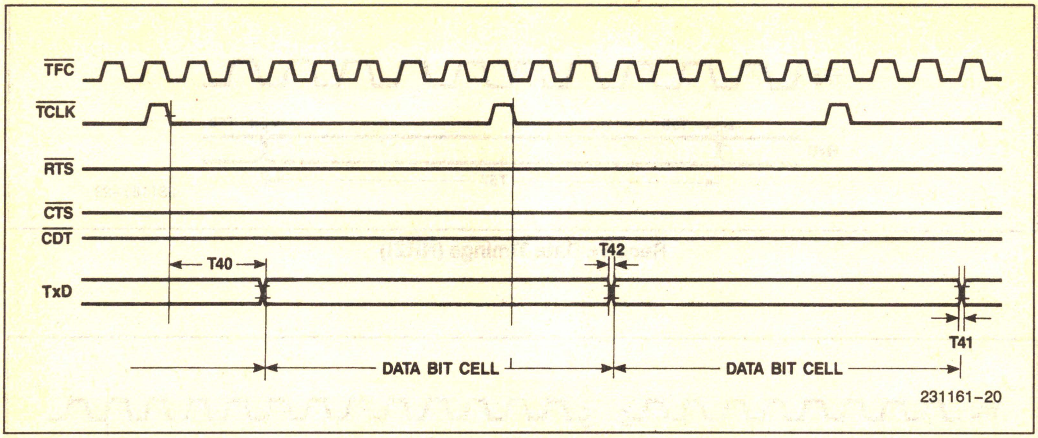
Transmit Timings: Clocks $\overline{RTS}$ and $\overline{CTS}$



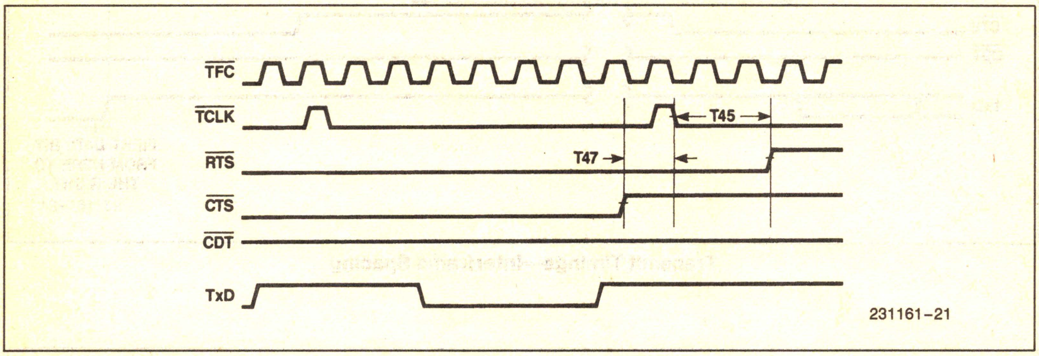
Transmit Timings—Manchester Data Encoding



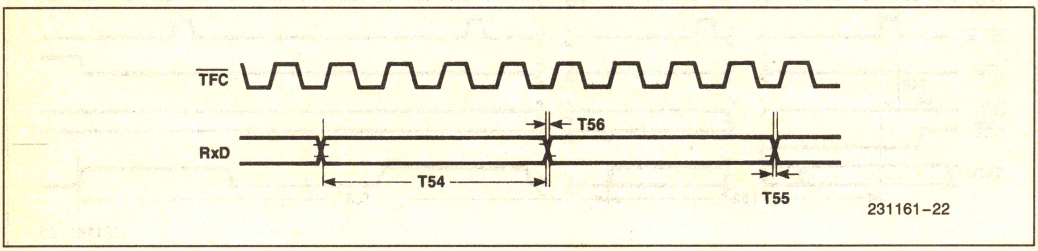
Transmit Timings—Lost $\overline{CTS}$



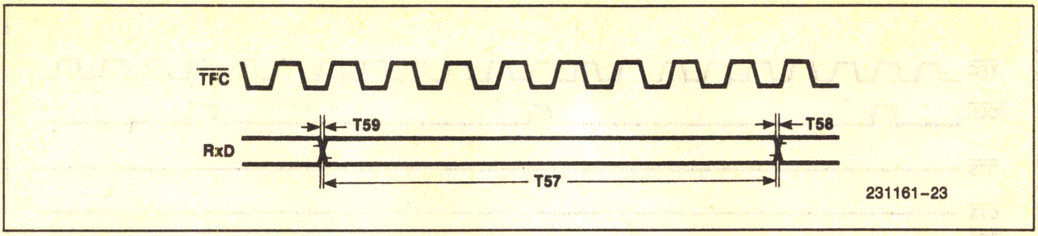
Transmit Timings—NRZI Data Encoding



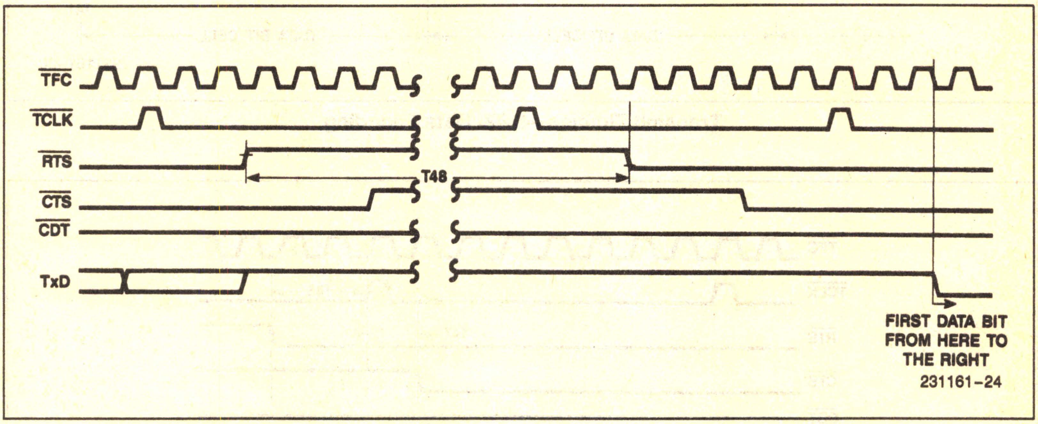
Transmit Timings—Lost $\overline{CTS}$



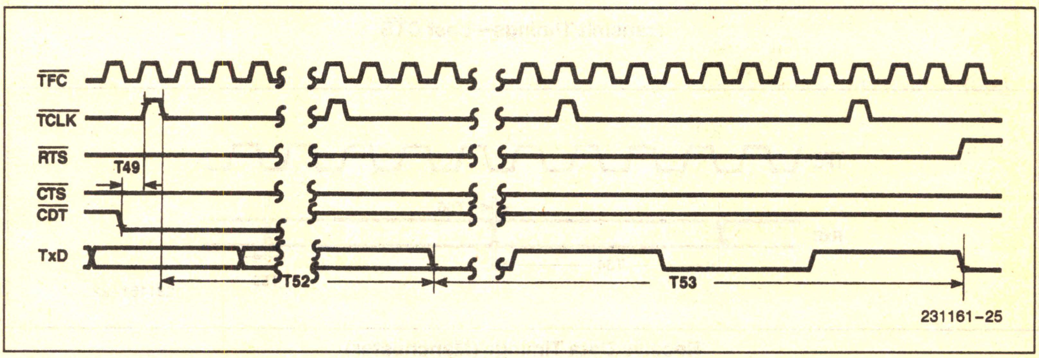
Receive Data Timings (Manchester)



Receive Data Timings (NRZI)



Transmit Timings—Interframe Spacing



Transmit Timings—Collision Detect and Jamming

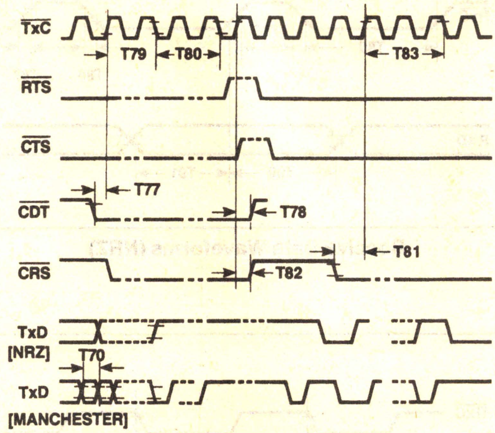
High Speed Mode

Symbol	Parameter	Min	Max	Units	Test Conditions
Transmit/Receive Clock Parameters					
T60	$\overline{\text{RxC}}$ $\overline{\text{TxC}}$ Cycle	200	*13	ns	
T61	$\overline{\text{TxC}}$ Rise Time		10	ns	*1
T62	$\overline{\text{TxC}}$ Fall Time		10	ns	*1
T63	$\overline{\text{TxC}}$ High	80	1000	ns	*1, *3
T64	$\overline{\text{TxC}}$ Low	80		ns	*1, *3
Transmit Data Parameters					
T65	TxD Rise Time		20	ns	*4
T66	TxD Fall Time		20	ns	*4
T67	$\overline{\text{TxC}}$ Low to TxD Valid		60	ns	*4, *6
T68	$\overline{\text{TxC}}$ Low to TxD Transition		60	ns	*2, *4
T69	$\overline{\text{TxC}}$ High to TxD Transition		60	ns	*2, *4
T70	TxD Transition—Transition	70			*2, *4
T71	$\overline{\text{TxC}}$ Low to TxD High (At the Transmission End)		60	ns	*4
$\overline{\text{RTS}}$, $\overline{\text{CTS}}$ Parameters					
T72	$\overline{\text{TxC}}$, Low to $\overline{\text{RTS}}$ Low Time to Activate $\overline{\text{RTS}}$		60	ns	*5
T73	$\overline{\text{CTS}}$ Low to $\overline{\text{TxC}}$ Low $\overline{\text{CTS}}$ Setup Time	65		ns	
T74	$\overline{\text{TxC}}$ Low to $\overline{\text{RTS}}$ High		60	ns	*5
T75	$\overline{\text{TxC}}$ Low to $\overline{\text{CTS}}$ Invalid	20		ns	
T75A	$\overline{\text{CTS}}$ High to $\overline{\text{TxC}}$ Low $\overline{\text{CTS}}$ Set-up Time to Stop Transmission	65		ns	*7
Interframe Spacing Parameters					
T76	Inter Frame Delay	*9		ns	
$\overline{\text{CRS}}$, $\overline{\text{CDT}}$, Parameters					
T77	$\overline{\text{CDT}}$ Low to $\overline{\text{TxC}}$ High External Collision Detect Setup Time	45		ns	
T78	$\overline{\text{TxC}}$ High to $\overline{\text{CDT}}$ Inactive $\overline{\text{CDT}}$ Hold Time	20		ns	*14
T79	$\overline{\text{CDT}}$ Low to Jamming Start		*10	ns	
T80	Jamming Period	*11		ns	
T81	$\overline{\text{CRS}}$ Low to $\overline{\text{TxC}}$ High Carrier Sense Setup Time	45		ns	*14
T82	$\overline{\text{TxC}}$ High to $\overline{\text{CRS}}$ Inactive $\overline{\text{CRS}}$ Hold Time	20		ns	*14

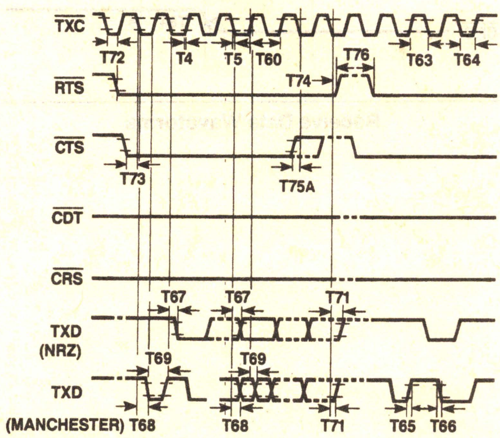
High Speed Mode (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
$\overline{CRS}$, $\overline{CDT}$, Parameters (Continued)					
T83	$\overline{CRS}$ High to Jaming (Internal Collision Detect)		*12	ns	
T84	$\overline{CRS}$ High to $\overline{RxC}$ High. End of Receive Packet	80		ns	
T85	$\overline{RxC}$ High to $\overline{CRS}$ High. End of Receive Packet.	20		ns	
Receive Clock Parameters					
T86	$\overline{RxC}$ Rise Time		10	ns	*1
T87	$\overline{RxC}$ Fall Time		10	ns	*1
T88	$\overline{RxC}$ High Time	80		ns	*1
T89	$\overline{RxC}$ Low Time	80		ns	*1
Received Data Parameters					
T90	RxD Setup Time	45		ns	*1
T91	RxD Hold Time	45		ns	*1
T92	RxD Rise Time		20	ns	*1
T93	RxD Fall Time		20	ns	*1

NOTES:
*1 — MOS levels.
*2 — Manchester only.
*3 — Manchester. Needs 50% duty cycle.
*4 — 1 TTL load + 50 pF.
*5 — 1 TTL load + 100 pF.
*6 — NRZ only.
*7 — Abnormal end to transmissions: $\overline{CTS}$ expires before $\overline{RTS}$.
*8 — Normal end to transmission.
*9 — Programmable value.
T76 = NIFS × T60 (ns)
NIFS - the IFS configuration value.
If NIFS is less than 12, then NIFS is enforced to 12.
*10 — Programmable value:
T79 = NCDF × T60 + (12 to 15) × T60 (ns) (if collision occurs after preamble).
*11 — T80 = 32 × T60
*12 — Programmable value:
NCSF × TTRC + (12 to 15) × TTRC
T83 = NCSF × T60 + (12 to 15) × T60
NCDF - collision detect filter configuration value.
*13 — 2000 ns if configured for Manchester encoding.
*14 — To guarantee recognition on the next clock.

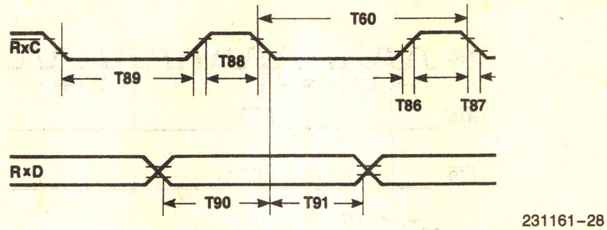


231161-26

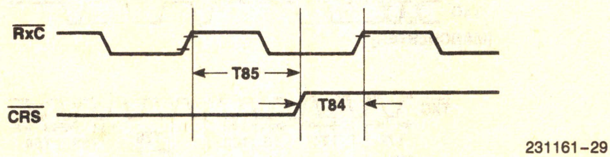


231161-27

Transmit Data Waveforms



Receive Data Waveforms (NRZ)



Receive Data Waveforms

CHAPTER 1
INTRODUCTION

OVERVIEW

Through the 1970's, with the evolution of the microprocessor, the concept of distributed processing became ever more attractive. The ability to integrate the intelligence of yesterday's boards onto todays chips drove the cost of computing to a point where decentralization of processing was not only feasible, but practical as well. In the 1980's, we have seen the advent of engineering workstations, instrumentation, personal computers, printers, and much more, all with the capability of performing their respective tasks independently of any supervisory machine. This decentralization of computing power has brought about the need to interconnect the different "nodes" in order to fulfill the following requirements. First, by interconnecting, we allow for the sharing of expensive peripheral resources like printers, plotters and file servers. Second, we allow for the sharing of information via common data bases. Finally, high level services, such as electronic mail, are supplied. Thus was the concept of networking born.

The definition of a network in its broadest sense is a system of processing units connected by communication lines. This definition can be broken down further based on the characteristics of a specific network. One special type of network is a Local Area Network or LAN. A LAN is defined as being a network supporting peer-to-peer communication over distances of tens of meters to several kilometers. Peer-to-peer implies that each station on the network is its own boss, i.e., there is no master-slave relationship. This chapter describes Local Area Networks vis-a-vis open system architecture. Also, a discussion of existing and emerging Local Area Network standards is presented, followed by an overview of Intel's Local Area Network components and related products and how they map into the existing and emerging standards.

LAN Requirements

Figure 1 depicts a Local Area Network model which is becoming representative of the office automation environment. Tier 1 consists of departmental clusters linking, for example, a group of PCs in the Finance department or several process control stations on the factory floor. Tier 2 provides the interface between various computers and departmental clusters, plus interconnection to system resources such as file servers, print servers, and communications servers.

Just as the different tiers have varied requirements, so, too, will the characteristics of each department's LAN vary depending on which organization they serve within the company. In general, the different departments within most companies can be divided into three distinct environments: factory, engineering, and office. On the factory floor, where a typical application is networking process control stations and robotics controllers, a network must be able to span a large distance while maintaining a high degree of noise immunity. Futhermore, the ability for each node to send a message must be deterministic, in that each must have a chance to transmit in a given interval of time. For engineering applications, in which CAD/CAE workstations are interconnected, the required network characteristics are high throughput over short to moderate distances. Finally, in the office the primary need is the interconnection of personal computers and servers. This application is highly cost sensitive but doesn't need the same high throughput or distance as in the factory. Because of the different applications and varied requirements of each, no single LAN can meet all needs. As such, many LANs have evolved over the last five to ten years.

There are four attributes that a Local Area Network must have in order to insure its own longevity. First, it must be backed by one or more major companies. Not only will the associated companies add credibility to the network, but they also provide the capital to cover development costs. Second, the existence of VLSI is essential. VLSI reduces cost and simplifies the design. Standardized software is the third essential ingredient; it allows for standard interfaces in addition to minimizing time-to-market. Fourth, the network must be easy to install and manage. Finally, and most importantly, the network itself must be an industry standard. The end user needs to be able to purchase a personal computer from one company, a file server from a second and a printer from a third, interconnect them by simply "plugging them in" and not have to worry about low level concerns such as protocols or media.

The International Standards Organization (ISO) and the IEEE have for several years been developing the models on which networking standards are based as well as the standards themselves. The next section overviews the activities of these two bodies in generating networking standards.

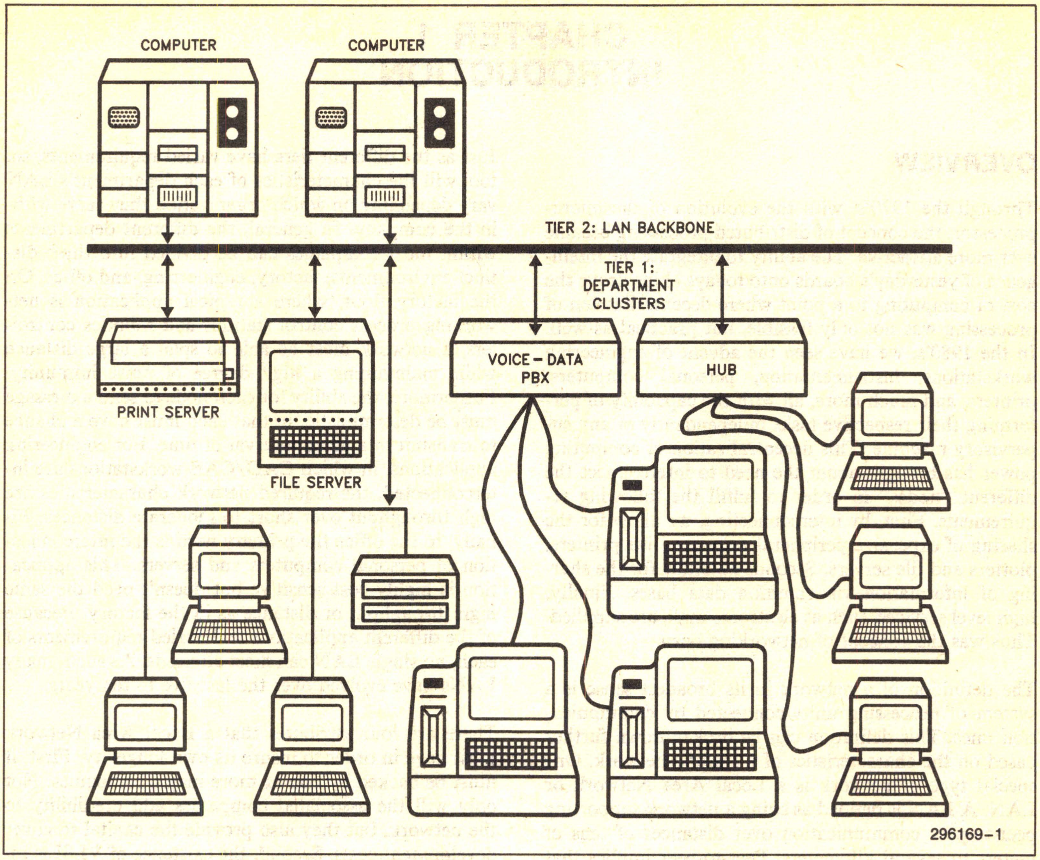


Figure 1. LAN Model

1.0 NETWORKING SOLUTIONS VIA STANDARDS

Networking is not a new concept, but until fairly recently, networks were of a proprietary nature. In the past, an end user had to buy all of its computing equipment from the same vendor in order to interconnect it. This problem was solved when the concept of "open systems" was developed. An open system is built using widely accepted standards. Open systems allow the end user to purchase equipment from several vendors in order to realize an optimal solution for any given application. Note that when a standard becomes widely accepted, it also becomes feasible to implement the required logic in VLSI, thereby lowering the connection cost.

1.1 The OSI Reference Model

The International Standard Organization (ISO), in an effort to encourage "open" networks, developed the Open Systems Interconnect (OSI) Reference Model. In simple terms, the model logically groups the functions and sets of rules, called protocols, necessary to establish and conduct communication between two or more parties. The model consists of seven sets of functions, often referred to as layers. The OSI model describes the functions of each later in broad terms, not specific implementations.

This layered model approach affords two key advantages. First, layers allow a clear division of the design task through modularity. Second, systems based on a layered architecture are flexible. Flexibility is achieved because each layer functions independently of the layer above or below it. Thus, specific layer implementations can be changed easily. For example, layers 1 and 2 of a network can be changed to be either CSMA/CD based (e.g., IEEE 802.3) or token ring based (e.g., IEEE 802.5), without affecting layers 3 through 7.

The layer functions of the OSI model are summarized in Figure 2.

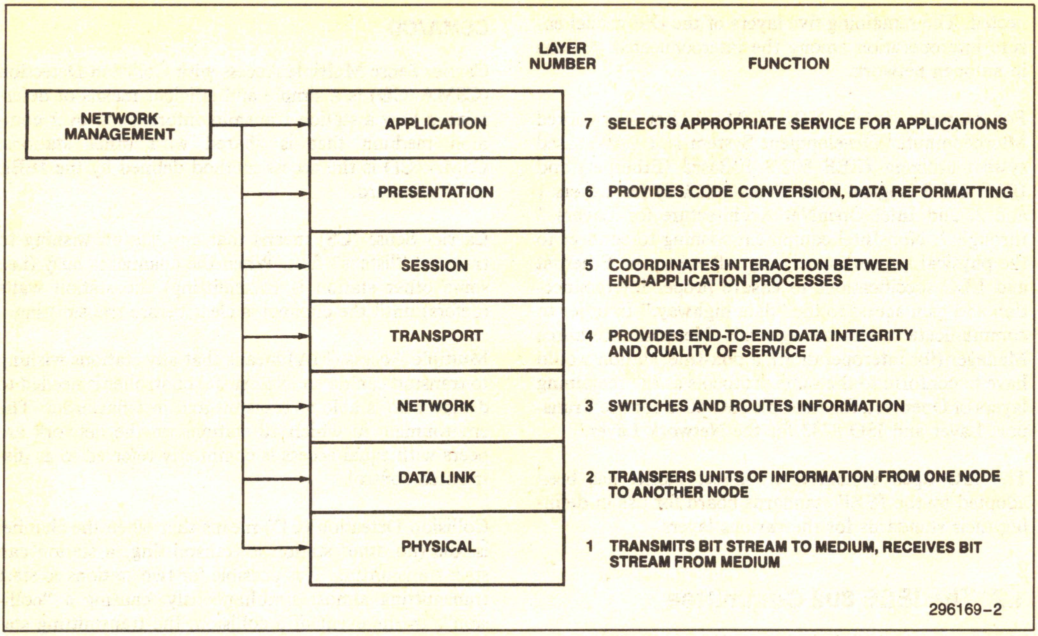


Figure 2. Layer Functions of the OSI Model

The **Physical Layer** describes the physical media over which the bit stream is to be transmitted. This Layer specifies type of cable (coax, twisted pair, etc.), connectors, signal levels, bit rate, data encoding method, modulation method, and method for detecting collisions in contention networks. In short, this Layer describes the actual physical media over which the bit stream is transmitted and the method of transmission, i.e., baseband or broadband.

The **Data Link** describes the rules for transmitting on the channel (made up of the encoder/decoder, transceiver cable, and transmission medium). Such items as the format of the information (frame) and procedures for gaining control of the channel (access method), transmitting the frame and releasing the physical media are specified by the Data Link Layer.

The **Network Layer** controls switching between links in a multihop network. The Network Layer is not necessary for a single LAN system because all stations connected onto a LAN share the same channel. This Layer is critical in gateway, communication server, and dial-up-communication applications.

The **Transport Layer** ensures end-to-end message integrity and provides for the required quality of service for exchanged information. For example, end-to-end acknowledgements and flow control are performed by the Transport Layer.

The **Session Layer** establishes and terminates logical connections between network entities. This Layer is also responsible for the mapping of logical names into network addresses.

The **Presentation Layer** provides for any necessary translation, format conversion, or code conversion to put the information into a recognizable form.

The **Application Layer** provides network based services to the end user. Examples of network services are distributed data bases and electronic mail. The Application Layer is not to be confused with the end user application itself.

Network Management is responsible for operation planning, which includes the gathering of operational statistics such as errors and traffic. It is also responsible for network initialization and maintenance (fault isolation). Network Management interfaces to each of the seven layers.

THE OSI MODEL AND NETWORK IMPLEMENTATIONS

The Physical and Data Link Layers of the OSI model ensure interconnectability. By implementing a particular physical and data link specification, equipment from multiple vendors can be physically and electrically con-

nected. The remaining five layers of the OSI model ensure interoperation among the interconnected stations in an open network.

For example, Intel's NDS-II Multi-User Networked Microcomputer Development System is a LAN based system utilizing IEEE 802.3 10Base5 (Ethernet) and IEEE 802.2 Logical Link Control (LLC) for Layers 1 and 2, and Intel OpenNet Architecture for Layers 3 through 7. Non-Intel equipment wishing to connect to the physical network need only adhere to the Ethernet and LLC specifications to ensure proper interconnection and gain access to the "data highway." In order to communicate with the system's Network Resource Manager (for interoperation), a non-Intel station would have to conform to the same protocols as the remaining layers of OpenNet, for example ISO 8043 for the Transport Layer and ISO 8743 for the Network Layer.

The ISO open system interconnect model has been adopted by the IEEE standards board for use in defining their standards for the various layers.

1.2 The IEEE 802 Committee

The Institute of Electrical and Electronics Engineers (IEEE), in response to a need for standardization in the field of Local Area Networks, formed the IEEE 802 standards body. The 802 standards specify the different protocols, access methods and their relationship with the ISO Open System Interconnection (OSI) Reference Model for Layer 1 and Layer 2 (see Figure 3). IEEE 802.1 explains how the different standards relate to each other and how they map into the OSI model. The Logical Link Control standard is specified by IEEE 802.2. IEEE 802.3, 802.4 and 802.5 specify the three different media access methods; CSMA/CD, Token Passing Bus and Token Ring, respectively.

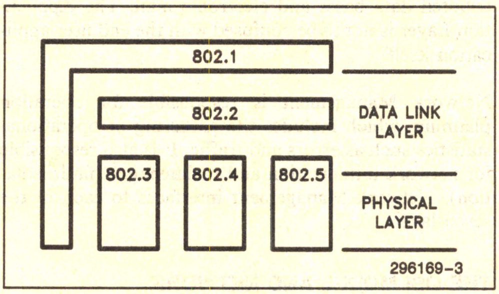


Figure 3. The IEEE 802 Standards Committees

CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a simple and efficient means of determining how a station transmits information over common medium that is shared with other stations. CSMA/CD is the access method defined by the IEEE 802.3 standard.

Carrier Sense (CS) means that any station wishing to transmit "listens" first. When the channel is busy (i.e., some other station is transmitting) the station waits (defers) until the channel is clear before transmitting.

Multiple Access (MA) means that any stations wishing to transmit can do so. No central controller is needed to decide who is able to transmit and in what order. The environment in which all stations on the network are peers with equal access is commonly referred to as distributed control.

Collision Detection (CD) means that when the channel is idle (no other station is transmitting) a station can start transmitting. It is possible for two stations to start transmitting almost simultaneously, causing a "collision". In the event of a collision, the transmitting stations will continue transmitting for a fixed time to ensure that all transmitting stations detect the collision. This is known as jamming. After the jam, the stations stop transmitting and wait a random period of time before retrying. The range of random wait times increases with the number of successive collisions such that collisions can be resolved even if a large number of stations are colliding.

There are three significant advantages to the CSMA/CD protocol. The first and foremost is that CSMA/CD is a proven technology. One CSMA/CD network, Ethernet, has been used by Xerox since 1975. Ethernet is so well understood and accepted that IEEE adopted it (with minor changes) as the IEEE 802.3 type 10BASE5 (10 Mbps, Baseband, 500 meters per segment) standard. Reliability is the second advantage to the 802.3 protocol. This media access method enables the network to operate without central control or switching. Thus, if a single station malfunctions, the rest of the network can continue operation. Finally, since CSMA/CD networks are of a passive, distributed nature, they allow for easy expansion. New nodes can be added at any time without reinitializing the entire network.

IEEE 802.3 Type 10BASE2 (about 200 meters per segment) is a variant which uses lower cost coaxial cable, and is more oriented to installation by the user. Popularly known as Cheapernet, 10BASE2 also eliminates a rather costly cable interconnecting the computer equipment to the transceiver which attaches to the coaxial cable in the case of 10BASE5.

TOKEN PASSING BUS

The IEEE 802.4 standard outlines the Token Bus media access method. A token is a group of coded bits that are passed from station to station on the network. The station that controls or "possesses" the token also controls the right of access to the link. So, as the token is passed, control of medium access is also passed. The link is a physical bus in that all the stations are attached to a common medium, but the token is passed from station to station forming a logical ring. For this reason, the token bus protocol has been identified as being a "logical ring on a physical bus".

During steady state operation, there are two general activities, data transfer and token passing. During the data transfer phase, a given station possesses the token and simply transmits data to one or more of the other stations of the link. Stations not possessing the token do not transmit, but listen to transmission for an address match. After a transmission is complete, the token is passed to the next station in the logical ring. This is the token passing phase.

Although the transmission of data and token passing are straightforward operations, network maintenance tasks make the token passing bus protocol relatively complex. The functions of network initialization, lost token recovery, adding new stations and general logical ring housekeeping require substantial logic. Additionally, all these functions must be replicated among all the token using stations on a given network in a prioritized fashion.

The primary advantage of token passing protocols is that they are deterministic as opposed to probabilistic. Every station has a definite opportunity to transmit within a given interval of time. With CSMA/CD, there is a statistical probability that each station will be able to transmit. In addition to the deterministic feature, token bus networks have the ability to span large distances, more than 10 kilometers.

TOKEN RING

Token ring is similar to token bus except that in addition to being a logical ring it is also a physical ring. The token ring procotol is governed by the IEEE 802.5 specification. It has similar performance characteristics as the token bus protocol.

1.3 Existing and Emerging Medium Access Standards

It has been established that no single LAN technology can satisfy the needs of every application. But, at the same time, the LAN marketplace cannot support every protocol that comes along. As a result, there are certain medium access control technologies that will be accepted as industry standards for given applications. Described here are the protocols that are already accepted, as well as those that are emerging, as industry-wide standards.

ETHERNET/CHEAPERNET

The IEEE 802.3 10BASE5 standard (Ethernet) has gained wide acceptance by both large and small corporations as a high speed (10 Mbps) Local Area Network. The Ethernet channel is a low noise, shielded 50Ω coaxial cable over which information is transmitted at 10 million bits per second. Each segment of cable can be up to 500 meters in length, can support up to 100 nodes, and can be connected into longer networks using repeaters. Repeaters are responsible for regenerating the signal from one cable segment on another. At each end of a cable segment a terminator is attached. This passive device provides the proper electrical termination to eliminate reflections.

The transceiver transmits and receives signals on the coaxial cable. In addition, it isolates the node from the channel so that a failure within the node will not affect the rest of the network. The transceiver is also responsible for detecting when two or more stations transmit simultaneously (collisions). Ethernet transceivers are connected to the network coaxial cable using a simple tap, and to the station it serves via the transceiver cable which can be up to 50 meters in length.

An Ethernet interface, which includes a serial interface and a data link controller, provides the connection to the user or server station. It also performs frame manipulation, addressing, detecting transmission errors, network link management and encoding and decoding of the data to and from the transceiver.

Due to its high speed and relatively high cost, Ethernet is targeted to be a LAN backbone for the engineering and office environments. It is also feasible to utilize Ethernet in engineering workstation and personal computer cluster applications where cost is not a sensitive issue.

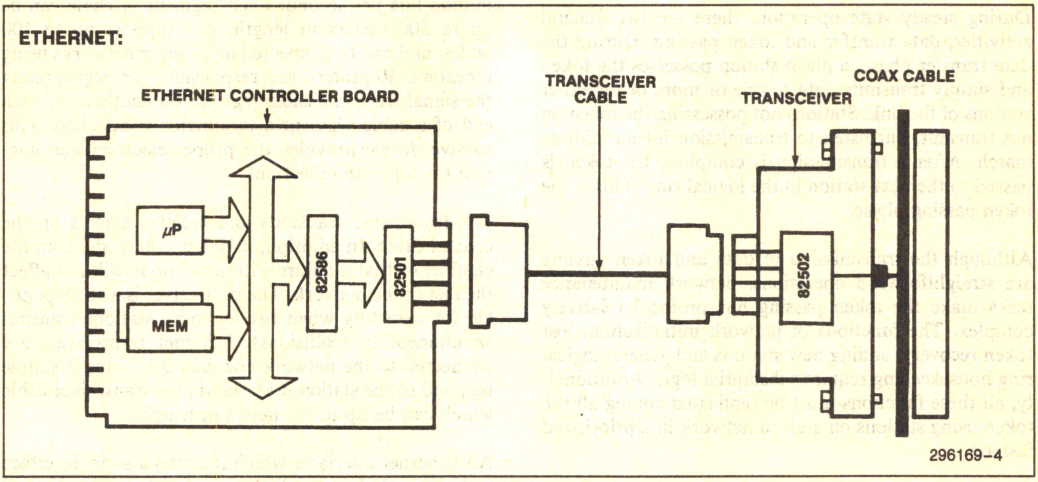
One of the major contributors to the high cost per connection of Ethernet is the cable. Ethernet cable must be highly immune to noise so that each cable segment can be 500 meters in length. Cheapernet was developed. Cheapernet, IEEE 802.3 IOBASE2, is identical to Ethernet with the following exceptions. First, less expensive RG-58 CATV coaxial cable is used as the network link. Second, the transceiver function is integrated into the node which is connected to the link via a BNC T-connector. Use of the lower performance cable limits Cheapernet to 185 meter segments with 30 nodes per segment. And finally, the transceiver cable is eliminated.

Cheapernet preserves the high data transfer rate of Ethernet, but reduces the per node cost considerably. For this reason, Cheapernet is optimized for high performance workstation or personal computer network clusters.

IBM PC NETWORK

In August 1984, IBM announced a new lost cost IBM PC Network based on 2 Mbps CSMA/CD broadband technology. The IBM PC Network uses standard CATV coaxial cables and hardware connections.

The IBM PC Network, in a user installable configuration, can handle up to 72 nodes and span 1000 foot radius. A key feature of this network is the IBM PC Network Translator Unit. The Translator Unit provides broadband frequency translation from the return channel to the forward channel, for a passive IBM PC Network. For greater capability, the IBM PC Network can be used in a professionally installed broadband network allowing up to 1000 IBM Personal Computers within a 5 Km radius of the network translator unit.



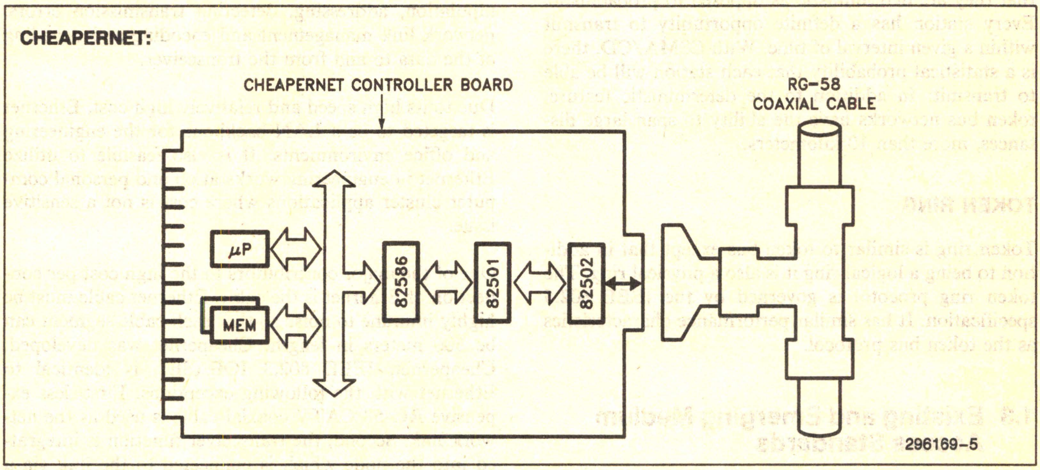


Figure 4. Ethernet/Cheapernet Configurations

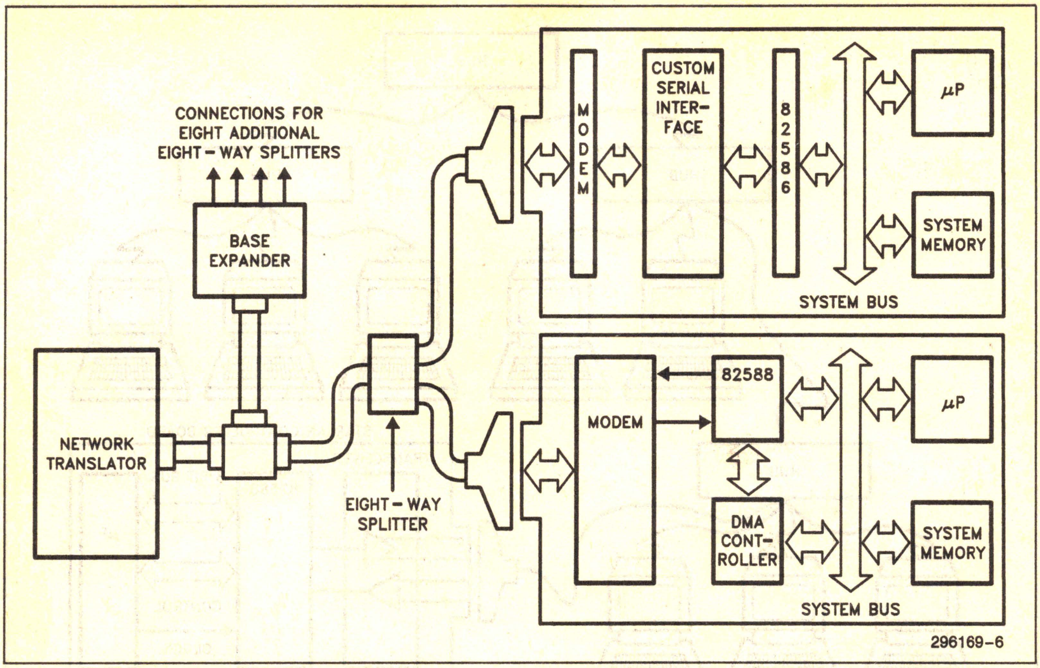


Figure 5. IBM PC Network Using Either the 82586 and the 82588 in Intelligent Adapters

StarLAN

IEEE 802.3 Type 1BASE5 (1 Mbps, Baseband, 500 meters diameter in a minimum configuration) is a CSMA/CD network oriented to personal computer interconnection in office environment. Each station is connected to a "hub" and each hub can be connected to another hub in point-to-point fashion. The result is cascaded star-shaped clusters. Each hub serves as the point of concentration, similar to a telephone wiring closet and performs two main functions: signal regeneration and retiming (for retransmission to other stations and hubs) and collision detection. When two stations transmit simultaneously and a collision occurs, the hub sends a collision presence signal to all receivers. The hubs can be cascaded up to five levels and each station-to-hub or hub-to-hub interconnection can be up to 250 meters in length.

StarLAN has two distinctive characteristics. Cost is the most sensitive issue in the personal computer field and the StarLAN hardware and cabling scheme are designed to be inexpensive. The cabling scheme uses standard twisted-pair telephone wiring and is laid out in a similar fashion where the StarLAN hub is analagous to a telephone wiring closet. Todays buildings are designed with this cabling scheme in mind which means installation, reconfiguring and servicing will be easy and low cost as well. Futhermore, most buildings use only about one half of the existing telephone cabling so the spares can be easily used for a StarLAN network.

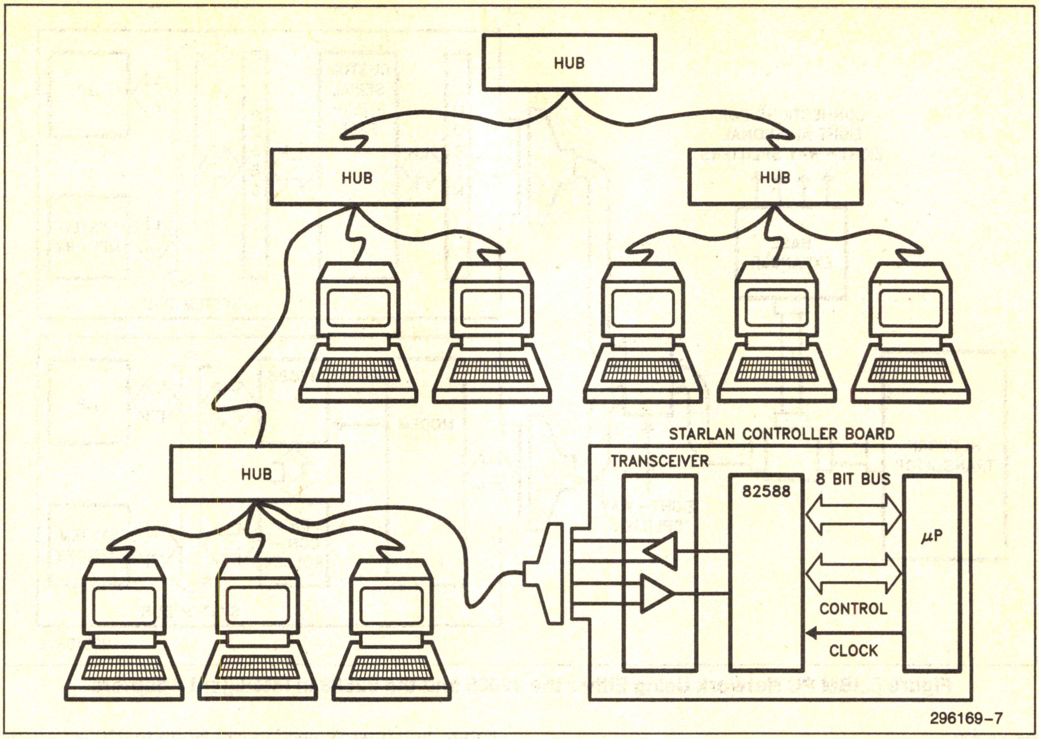


Figure 6. StarLAN Configuration

2.0 THE INTEL LAN SOLUTION

2.1 A Commitment To Standards

Intel has the most complete range of VLSI, software and boards to support the existing and emerging industry standards for Local Area Networks. The 82586 LAN Coprocessor, the 82C501 Ethernet Serial Interface and the 82502 Ethernet Transceiver Chip make up Intel's three chip set for IEEE 802.3 10 Mbps networks (10BASE5 and 10BASE2). For the mid-range LAN applications, i.e., StarLAN and the IBM PC Network, Intel offers the 82588 LAN Controller and the 82586 LAN Coprocessor. Finally, iNA960 software supports Intel hardware solutions with ISO conformant implementations. Intel also supplies several addition board and software solutions in the networking field. For information on these product lines, please contact your local Intel Field Sales representative.

2.2 Intel's Ethernet/Cheapernet Chip Set

THE 82586 LAN COPROCESSOR

The 82586 is an intelligent peripheral that completely manages the processes of transmitting and receiving frames over a CSMA/CD network. The 82586 offloads the host CPU of the tasks related to managing communication activities. The 82586's memory based architecture and integrated DMA controller enables it to function independently of the host CPU for time critical functions. Hence, the 82586 is truly a coprocessor.

In addition to the high performance, the 82586 also features a high degree of flexibility. The 82586's Network parameters are programmable so as to optimize the LAN design to specific applications. When powered up, the chip defaults to 10BASE5/10BASE2 network parameters. But when programmed accordingly, the 82586 can support StarLAN, the IBM PC Network, and a wide range of proprietary Local Area Networks. Because of its flexibility, the 82586 is also ideal for Serial Backplane applications.

The 82586 interfaces easily to available microprocessors. Systems requiring minimum component count can

take advantage of the 82586's direct interface (no "TTL glue") to Intel's 80188 (8-bit data bus) and 80186 (16-bit data bus) microprocessors.

The 82586 uses memory efficiently through data (or buffer) chaining. System memory is not wasted because short frames can be saved in buffers of minimal size, while long frames are saved by chaining buffers together. The 82586's memory based architecture, together with its data chaining capability result in a high degree of CPU independence, along with high system performance.

The 82586 provides a rich set of network management capabilities. Included are:

- Error tallies in system memory to monitor:
  - — Number of frames incorrectly received due to CRC errors
  - — Number of frames incorrectly received due to misaligned frames
  - — Number of collisions experienced in the process of transmitting frames
  - — Number of frames lost due to lack of receive buffers
  - — Number of frames lost due to DMA overrun while receiving frames
- Monitoring of Signal Quality Error (SQE) function as defined by the IEEE 802.3 specification (signaling that the transceiver's collision detection mechanism is not functioning correctly).

The 82586 provides diagnostic capability through internal and external loopbacks. Distance to cable breaks and shorts is provided by on-chip time domain reflectometry logic.

THE 82C501 ETHERNET SERIAL INTERFACE

The 82C501 is designed to work directly with the 82586 in 10 Mbps LAN applications. The primary function of the 82C501 is to perform Manchester encoding/decoding, provide the 10 MHz transmit to the 82586, recover clock from the received signal, generate carrier sense and collision presence signals appropriate to the 82586, and interface the transceiver cable. The 82C501 provides for fault isolation via an internal loopback. Continuous transmission (babbling) is prevented by an on-chip watchdog timer.

The 82C501 conforms to the IEEE 802.3 10BASE5 and 10BASE2 specifications. It is designed with Intel's CHMOS II technology.

THE 82502 ETHERNET TRANSCEIVER CHIP

The 82502 rounds out Intel's three chip set for IEEE 802.3 10 Mbps networks. The 82502's primary functions are transmission of data onto the network coaxial cable, reception of data from the coax, collision detection, and interface the transceiver cable. Additional features are:

- A jabber mechanism, namely a watchdog timer to prevent continuous transmission by the parent station.
- A defeatable Signal Quality Error (SQE) test which verifies functionality of the collision detection circuitry.
- On-chip precision voltage reference which allows for relaxed power supply tolerances.
- CHMOS technology which allows the 82502 to run at very low power consumption levels, thus enhancing reliability.

The 82502 supports 10BASE5, 10BASE2 and related repeater applications.

2.3 The 82588 LAN Controller

The 82588 is a highly integrated LAN controller targeted for use in cost sensitive CSMA/CD LAN applications, in particular, personal computer interconnection. The 82588 integrates CSMA/CD, Media Access Control (MAC) functions, data encoding/decoding, clock recovery from received signal, two different collision detection mechanisms and transmit clock generation. This high-integration allows the system designer to reduce component count, and thus reduce board space and development time. The functionality of the 82588 is optimized at up to 2 Mbps in either baseband or broadband networks.

Like the 82586, the network parameters of the 82588 are programmable, providing the flexibility to support numerous LAN applications. Among the programmable parameters are:

- — Slot Time
- — Minimum Frame Length
- — Framing (End-of-Transmission or HDLC)
- — Address Field Length
- — Interframe Spacing
- — NRZI, Manchester, or Differential Manchester encoding/decoding

Additionally, the 82588 supports various network topologies. The 82588 conforms to the IEEE 802.3 1BASE5 specification.

The 82588 implements two methods of logic based collision detection. The Code Violation method detects a collision when a data transition occurs outside of the specified regions. The Proprietary Bit Comparison method compares the "signature" of a transmitted frame to the receive frame "signature".

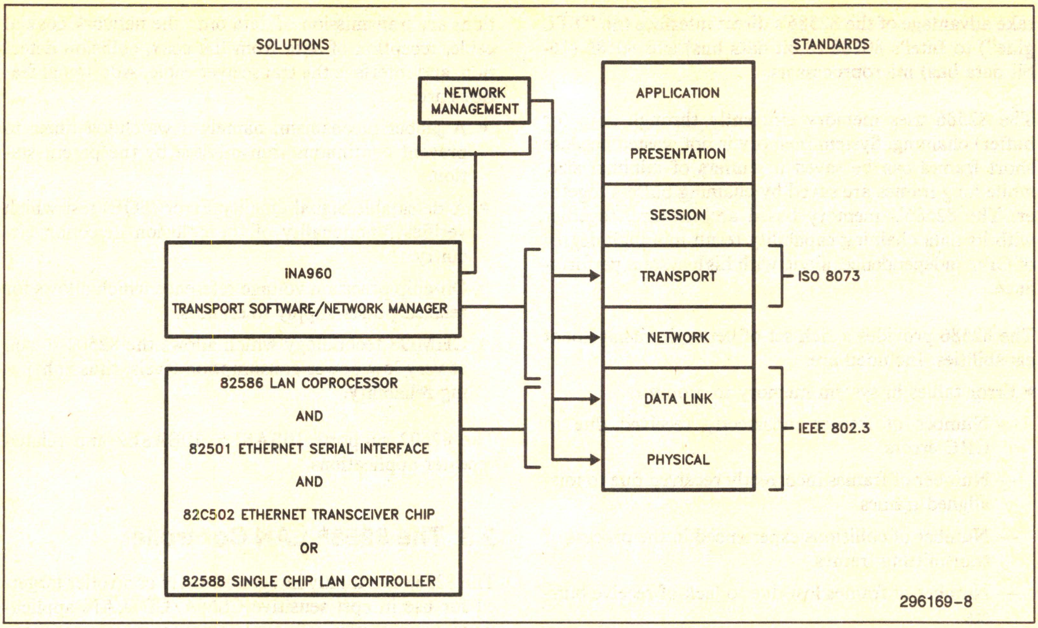


Figure 7. Intel's LAN Component Solutions

Another significant feature of the 82588 is its system interface. High level commands such as TRANSMIT and CONFIGURE are used. The 82588 supports receive buffer chaining, which provides for efficient memory usage. Finally, the 82588 has a complete set of network management, maintenance and diagnostic capabilities.

2.4 The iNA 960 Transport Software

iNA 960 is a general purpose Local Area Network software package that provides the user with guaranteed end to end message delivery. iNA 960 conforms to the International Standards Organization's 8073 specification for Class 4 Transport Layer services as well as ISO 8473 Network Layer services. iNA 960 also provides network management functions, along with 82586 and 82588 device drivers.

TRANSPORT SERVICES

The iNA 960 transport layer implements two kinds of message delivery services: virtual circuits and datagram. Virtual circuits provide a reliable point-to-point message delivery service ensuring maximum data integrity and are fully compatible with the ISO 8073 Class 4 protocol. In addition to guaranteed message integrity, iNA 960:

- Provides flow control (data rate matching between sender and receiver).
- Supports multiple simultaneous connections (process multiplexing).
- Handles variable length messages (independently of physical frame size).
- Supports expedited delivery (to transmit urgent data).

The datagram option provides "best effort" delivery service for non-critical messages. The datagram service does not guarantee message integrity but requires less channel overhead than virtual circuits.

NETWORK MANAGEMENT SERVICES

The Network Management facility supports the users of the network in planning, operating, and maintaining the network by providing network usage statistics, by allowing the monitoring of network functions and by detecting, isolating, and correcting network faults.

The Network Management facility also supports up-line dumping and down-line loading of data bases or to boot systems without local mass storage.

USER ENVIRONMENT

In the iRMX (Intel's real time, multitasking operating system) environment, both the user programs and iNA 960 run under iRMX 86. The communications software is implemented as an iRMX 86 job requiring the nucleus only for most operations. The only exception is the boot server option, which also needs the Basic I/O Sys-

tem. iNA 960 will run in any iRMX environment including configurations based on the 80130 software on silicon component.

In those systems where iRMX 86 is not the primary operating system, or where offloading the host of the communications tasks is necessary for performance reasons, the user may wish to dedicate a processor for communication purposes. iNA 960 can be configured to support such implementations by providing network services on an 8086, 8088, or 80186 microprocessor.

iNA 960 is also available for operation under MS DOS in personal computer environments.

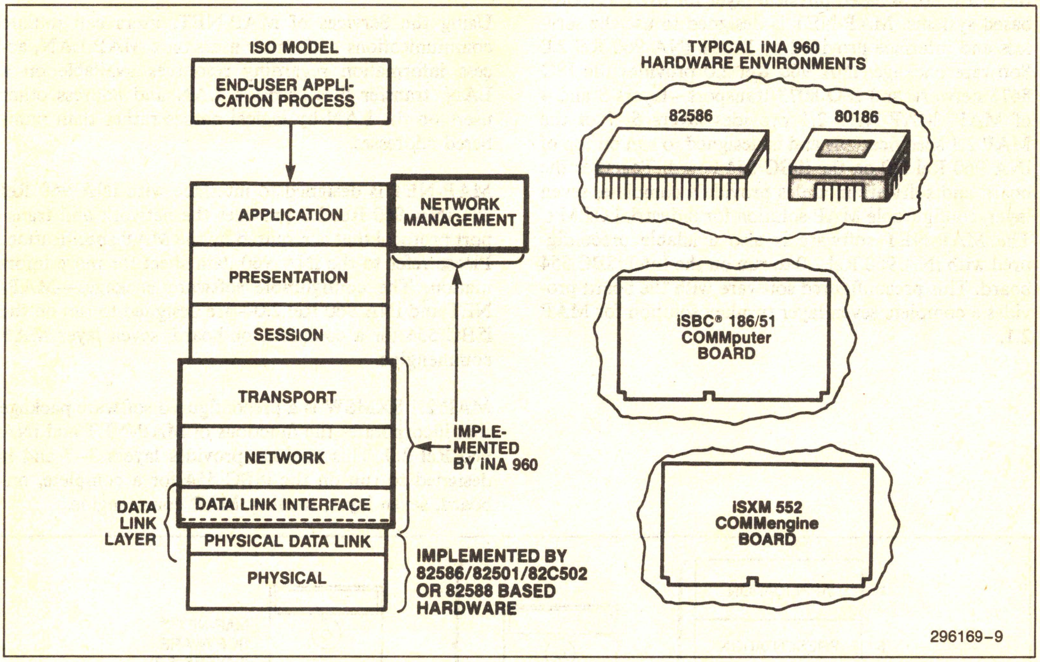


Figure 8. iNA 960 Software

2.5 MAP-NET Layer 5–7 Map Software

MAP-NET 2.1, iNA 960 Rel 2.0 and the iSBC 554 MAP Board and ready-to-use software building blocks for OEM suppliers of networked systems to implement ISO/OSI layers 1–7, as specified by MAP version 2.1. The Intel iSBC 554 board provides the data link and the IEEE 802.4 based physical layer for MULTIBUS® based systems. MAP-NET is designed to use the services and interface provided by Intel's iNA 960 Rel 2.0 Software package. iNA 960 Rel 2.0 provides the ISO 8473 network and ISO 8073 transport—layers 3 and 4 of MAP. MAP-NET 2.1 provides layers 5–7 of the MAP 2.1 specifications and is designed to run on top of iNA 960 Rel 2.0 on the iSBC 554 board. Together the board and software modules provide a complete, seven layer, configurable MAP solution for industrial OEM's. The MAP-NET software is also available preconfigured with iNA 960 Rel 2.0 to run on the Intel iSBC 554 board. This preconfigured software with the board provides a complete seven layer turnkey solution for MAP 2.1.

FUNCTIONAL OVERVIEW

The Intel MAP-NET 2.1 software provides the following services specified by MAP 2.1; the session service, network management, FTAM and CASE. These services fit into the upper 3 layers of the ISO/OSI 7 layer model.

Using the Services of MAP-NET, users can initiate communications with other users on a MAP LAN, access information regarding resources available on a LAN, transfer files across a LAN and address other users on the LAN by logical names rather than numbered addresses.

MAP-NET is designed to interface with iNA 960 Rel 2.0. iNA 960 Rel 2.0 provides the network and transport protocol that is required by the MAP specification. Please refer to the iNA 960 data sheet for more information. The configurable software packages—MAP-NET and iNA 960 Rel 2.0—are designed to run on the iSBC 554 for a complete, on-board, seven layer MAP commengine.

MAP 2.1 SXMSW is a preconfigured software package that incorporates the functions of MAP-NET and iNA 960 Rel 2.0. This package provides layers 3–7 and is designed to run on the iSBC 554 for a complete, on-board, seven layer, turnkey MAP commengine.

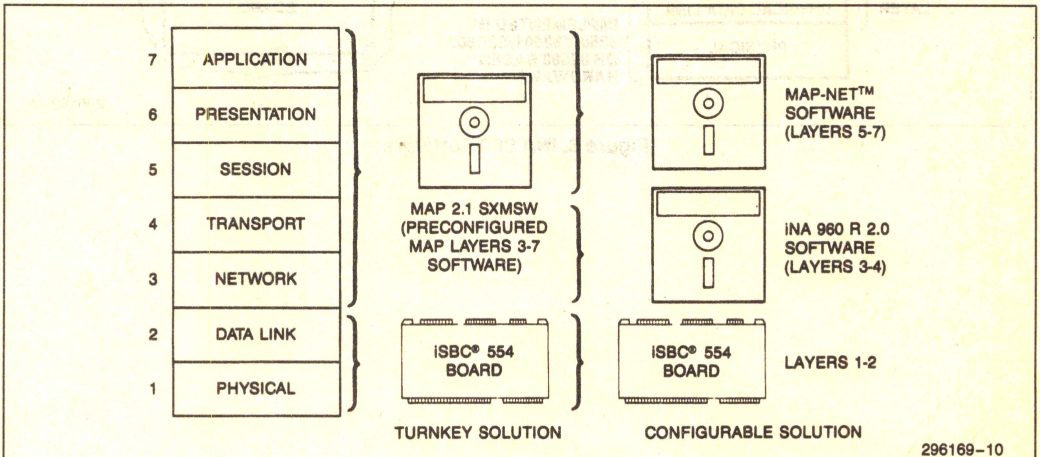


Figure 9. OSI Reference Model. Figure 9 Above Indicates How Intel's MAP Software and Hardware Products Fit In the OSI Reference Model for MAP.

CHAPTER 2
THE 82586 LAN COPROCESSOR

OVERVIEW

This chapter describes the features and operation of the 82586 LAN Coprocessor. It is assumed that the reader is familiar with the basic concepts of Data Communication, Local Area Networks (LAN), and the IEEE 802.3 Standard.

This Chapter is divided into three parts, each part covers several sections:

1) A general description of the 82586.
2) A description of the major functions performed by the 82586 for the user:
   - Transmit Functions
   - Receive Functions
   - Network Management and Diagnostic Functions
   - Interaction with the Host CPU
3) Detailed instructions on how to use the 82586:
   - Initializing the 82586
   - Controlling the 82586
   - Action Commands
   - Frame Reception
   - Bus Interface Hardware
   - Network Interface Hardware

Pin functions, electrical and timing characteristics are located in the 82586 Data Sheet, included as part of this User's Manual.

1.0 OVERVIEW OF THE 82586 LAN COPROCESSOR

The 82586 VLSI chip is an intelligent, high performance, CSMA/CD (Carrier Sense Multiple Access with Collision Detection) communications controller. The 82586 performs all functions associated with data transfer between user memory and the Network: framing, link management, address filtering, error detection, data encoding, network management, Direct Memory Access (DMA), buffer chaining, and interpretation of high level commands from the user. Called the LAN Coprocessor, the 82586 was designed to relieve the host CPU of most tasks associated with controlling access to a LAN.

The 82586 meets the performance requirements of the IEEE 802.3 Standard: 10 megabits per second bit rate and 9.6 microseconds Interframe Spacing. In addition to providing DMA transfers at 4 megabytes per second, it tolerates local bus latency of over 10 microseconds without losing data, and bus transfer rates as low as 2 megabytes per second. The high performance permits the 82586 to be used in distributed processing applications such as high speed resource sharing and interprocessor communications.

The 82586's programmable network parameters allows it to serve as controller for a wide range of CSMA/CD type LAN's. It is compatible with network specifications such as high service (broadband), high performance (short topologies) and low cost (1 Mbps) networks. Data rates less than 10 megabits per second are supported. Many parameters are configurable including all framing parameters (i.e. address length, End of Carrier or Bitstuffing frame boundary delineation, etc.), Slot Time and Interframe Spacing.

Network and station reliability is enhanced by built-in diagnostic aids, such as Time Domain Reflectometer (TDR), external and internal loopback, Transceiver integrity verification, internal register dump and a self test procedure.

The 82586 is contained in a 48-pin dual in-line package. Figure 1 shows the pin layout.

Signals in parentheses are available in Minimum mode.

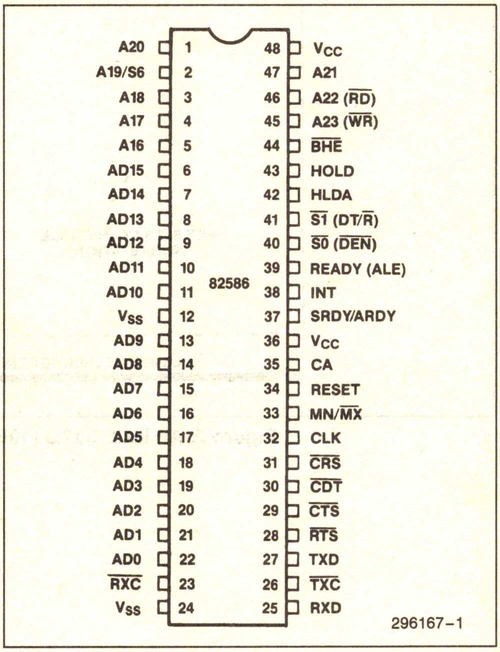


Figure 1. The 82586 Pin Configuration

The major application of the 82586 is to serve as the communication manager in a station connected to a LAN. Such a LAN station typically consists of a host CPU, shared memory, an 82586 Local Communication Controller, Serial Interface Unit, Transceiver, and LAN link. The 82586's task is to perform functions associated with transferring data between shared memory and the LAN link. As an example, Figure 2 shows the 82586 in a workstation connected to an IEEE 802.3 network.

The 82586 has two interfaces: Bus Interface to the local bus and the CPU; Network Interface to the Serial Interface Unit.

On the Bus side, the 82586 is a 'master' on the 8 or 16-bit local bus, and communicates directly with the CPU via Channel Attention (CA) and Interrupt (INT) signals. It is optimized for operation with the iAPX 186 bus but can be used with other general purpose processors.

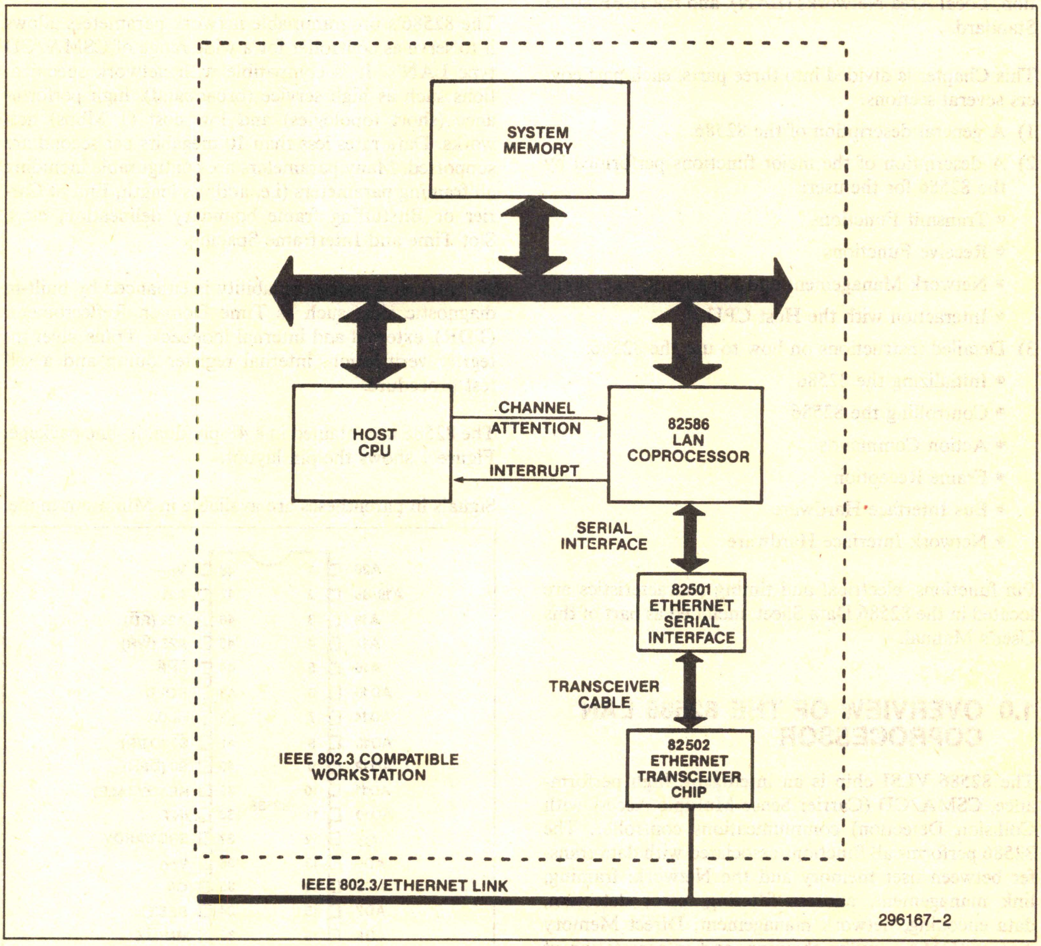


Figure 2. An IEEE 802.3 (10BASE5) Compatible Workstation

On the Network side, the 82586 is connected to an Ethernet Serial Interface that provides Transmit and Receive Clock and Data, Collision Detect, Carrier Sense, and Request to Send/Clear to Send signals to the 82586. The Ethernet Serial Interface is connected to the Transceiver, which is connected to the LAN link. In the particular case of the IEEE 802.3 (10BASE5) station, the Ethernet Serial Interface is Intel's 82501 and the Ethernet Transceiver is Intel's 82502.

2.0 82586 TRANSMIT FUNCTIONS

The 82586 LAN Coprocessor performs two major tasks: transmitting data from host memory to the Network and receiving data from the Network and placing it in memory. This section describes the transmission process. Reception is described in section 3.

The data units handled by the 82586 are frames. A frame is a sequence of bits that travels on the link. A frame is divided into fields: address, data, frame check sequence, etc. The host CPU prepares a sequence of frames in shared memory and instructs the 82586 to start transmission. Frames are transmitted by the 82586 one at a time. The chip resolves access and contention on the link using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Link Management mechanism.

This section presents the framing, link management, and priority mechanisms.

2.1 Framing

Framing has three primary functions: to determine the beginning and end of the frame (frame boundary delineation); to determine the frame's source and destination (addressing); and to perform error detection. Framing can be summarized in terms of the generalized frame format shown in Figure 3.

PREAMBLE
START FRAME DELIMITER FIELD
DESTINATION ADDRESS
SOURCE ADDRESS
LENGTH FIELD
DATA FIELD
FRAME CHECK SEQUENCE
END OF FRAME FLAG (OPTIONAL)
PADDING (OPTIONAL)

Figure 3. Frame Fields

Figure 3 shows the fields in a frame: the Preamble is used as a synchronizing sequence for bit decoding, followed by the Start Frame Delimiter Field, SFD. Next, the Destination Address is the frame target address. This field is followed by the Source Address (sender's address). The Length and Data Fields contain user supplied data. The Frame Check Sequence is a Cyclic Redundancy Check, CRC, used in detecting bit errors. Two optional fields may follow, End of Frame (EOF) flag and Padding. The latter extends the length of the frame to ensure minimum frame length.

The 82586 handles Frame Boundary Delineation completely transparent to the user. The fields involved in the Frame Boundary Delineation are: Preamble, SFD Field, EOF flag and Padding. The 82586 is configurable to one of two Frame delineation methods: End of Carrier or Bitstuffing.

In the End of Carrier method, the 82586 transmits (depending on the configuration) 8, 24, 56, or 120 Preamble bits of alternating ones and zeros followed by a SFD field (10101011). The end of frame is indicated by the carrier going inactive immediately after the Frame Check Sequence field. This frame boundary delineation method is compatible with IEEE 802.3.

The Bitstuffing method implements the HDLC zero bit insertion/deletion mechanism. The 82586 transmits (depending on the configuration) a Preamble of 8, 24, 56 or 120 alternating ones and zeros followed by an HDLC Flag (01111110). End of frame is indicated by an HDLC flag. The 82586 performs HDLC zero bit insertion (insert 0 after five consecutive 1's) on the fields between flags (exclusive). The chip can be configured to pad the frame with additional flags so that frame length becomes longer than Slot Time (see section 2.2).

Regardless of the Frame Boundary Delineation method, the two fields following the SFD Field are Destination Address and Source Address. The former is fetched from memory and the latter is usually inserted from the internal Individual Address register (unless configured not to). The address length can be configured to a value from 0 to 6 bytes. Addresses are sent with the least significant byte first. The Destination Address can be one of three types: Individual Address (least significant bit is zero), Multicast Address (least significant bit is one), or Broadcast (all ones). See section 3.2 for more details on addressing.

The Length Field and Data Field are fetched by the 82586 from memory and transmitted after the Source Address. The Length Field is 2 bytes long. The 82586 itself places no limit on the Data Field length.

The Frame Check Sequence field protects against bit errors in the frame. It is the result of a Cyclic Redundancy Check computed from the Destination Address, Source Address, Length Field, and Data Field. The chip can be configured to one of two CRC algorithms: the CCITT V.41 16-bit polynomial or the Autodin II (IEEE 802.3) 32-bit polynomial.

2.2 Link Management

The 82586 handles CSMA/CD link management algorithms according to the IEEE 802.3 standard. 82586 Link management algorithms are adaptable to a variety of network topologies via programmable configuration parameters. Station priorities are also programmable.

The 82586 constantly monitors link activity. Whenever it senses the carrier (data transitions) on the link, the 82586 defers to the passing frame by delaying any pending transmission. After carrier goes inactive, the 82586 continues to defer for Interframe Spacing time. Interframe Spacing is configurable from 32 to 255 TCLK (Transmit Clock) units. If, at the end of that time, it has a frame waiting to be transmitted, transmission is initiated independent of the sensed carrier.

After transmission has started, the 82586 attempts to transmit the entire frame. In the normal case, frame transmission is completed, and the host notified. Otherwise, one or more of the following events causes transmission to terminate prematurely: Clear-to-Send signal goes inactive during transmission, data transfer rate from memory to the chip did not keep up with transmission (DMA underrun), Carrier Sense goes inactive (in case the chip is configured to expect the return of Carrier Sense signal), a collision is detected via Collision Detect, or the collision retry counter exceeded the maximum specified value.

When the 82586 has finished deferring and has started transmission, it is still possible to experience link contention. This situation is called a collision and it is generally detected by the Transceiver. The 82586 enforces a collision by transmitting a Jam pattern of 32 ones. If a collision is detected during the Preamble, transmission of the Preamble is completed before jamming starts.

The dynamics of collision handling are largely determined by the Slot Time. Slot Time is the maximum end to end round trip delay time of the network plus jam time. Slot time is important because it is the worst case time to detect a collision. Long networks have longer slot times than short networks. Thus, the 82586 provides for slot time to be configurable from 1 to 2048 TCLK units.

After waiting the Backoff time, the 82586 attempts to retransmit the frame, unless the number of retransmissions has exceeded the maximum allowed. The 82586 calculates the Backoff algorithm according to the IEEE 802.3 standard: Backoff is an integral number of Slot Times. It is a random number, from 0 to maximum. The maximum number is (2**R)-1, where R is the minimum between 10 and the number of retransmission attempts. This range can be extended using Accelerated Contention Resolution mechanism, see section 2.3 and 2.4. The beginning of Backoff time is configurable to one of two methods: if configured to the IEEE 802.3 compatible method, it starts immediately after the end of jamming; if configured to the alternate Backoff method (designed for lower bit rates and/or shorter topologies), Backoff starts after the deferring period following collision.

The 82586 maintains a retry counter that is incremented after each retransmission attempt. If retransmission is successful, the user is notified. If the number of retries exceeds the maximum, an error is reported. The number of allowed retries is configurable from 0 to 15 attempts. The only difference between transmission and retransmission is that transmission clears the retry counter and retransmission increments it.

On completion of transmission or retransmission, the 82586 reports the number of collisions that occurred and whether it exceeded the maximum. It also indicates if the chip had to defer to passing traffic on its first transmission attempt.

The user may attempt to abort transmission. Upon receipt of the Abort command, the 82586 transmits a Jam pattern to cause a CRC mismatch. The chip reports to the host either that the abort succeeded or that the frame transmission completed before the abort was accepted.

2.3 Priority Mechanism

One of the goals for the IEEE 802.3 standard is to ensure fairness among stations trying to access the link. However, there are applications (e.g. transmitting voice) where station priority improves the performance. The 82586 implements two priority mechanisms: linear and accelerated contention resolution.

Linear priority determines the number of Slot Times the 82586 waits after deferring or the end of Backoff (whichever comes last) before transmitting. If the link becomes busy during the wait period, the process of deferring and waiting starts again. Linear priority is programmable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

Accelerated contention resolution extends the range from which the random number for Backoff is drawn. It is configurable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

2.4 Details of the Link Management Algorithms

The 82586 supports the IEEE 802.3 link management algorithms. In addition, the 82586 provides alternative station priority and backoff mechanisms.

The transmit link management includes two cooperating sequences: Frame Transmitter and Deference. The following variables provide the communication between these sequences: DEFERRING and WAITING from the Deference sequence and BACKOFF from the Frame Transmitter sequences.

Deference is an ever running sequence and performs the following algorithm compatible with IEEE 802.3:

1) Waits for Carrier Sense to become active.

2) Sets the DEFERRING variable.

3) Waits for Carrier Sense to become inactive.

4) Waits for a period of Interframe Spacing.

5) Clears the DEFERRING variable, waits for pending frame and returns to step 1.

The Frame Transmitter sequence is initiated when the 82586 is instructed to transmit a frame. It runs in parallel with Deferring and performs the following:

1) Assembles the frame and clears RETRIES.

2) Waits for DEFERRING and WAITING to clear.

3) Starts frame transmission.

4) If transmission is completed without detecting a collision, the sequence is completed with a successful transmission.

5) If a collision is detected, the 82586 does the following:

   a. Completes transmission of the Preamble (if applicable).

   b. Transmits Jam pattern.

   c. If RETRIES is equal to the maximum number of retries, the Transmit command is completed and status posted in the Transmit command status field.

   d. RETRIES is incremented if it was less than the maximum number of retries.

   e. The Backoff time is computed based on the following algorithm:

   Backoff time = (slot time) × (random number)

   Where:

   Random Number R is given by

   $0 \leq R < 2 \exp [\min (10, \text{retries})]$

   f. The 82586 waits for the period of Backoff time. If the Backoff time is zero (R = 0), then the Backoff period is equal to Interframe Spacing. For a random number greater than zero, Backoff is computed as shown in 'e' above.

   g. If Backoff time was equal to Interframe Spacing (R = 0), then the 82586 transmits after the Backoff time has expired.

   h. If random number is greater than 0, the 82586 waits for the Backoff time and returns to step 2.

   i. Clears the BACKOFF variable and returns to step 2.

LINEAR PRIORITY

In the case where the 82586's linear priority option has been selected, the link management algorithm differs in the deferring sequence:

1) Waits for Carrier Sense to become active.

2) Sets deferring variable.

3) Waits for Carrier Sense to become inactive.

4) Waits for a period of Interframe Spacing.

5) Calculates Wait Time as follows:

   Wait Time = P × Slot Time

   Where P is the programmed linear priority number

6) Sets the waiting variable.

7) Waits according to Wait Time calculating in step 5. If during the Wait Time, Carrier Sense goes active, the 82586 sets the deferring variable, clears the WAITING variable and returns to step 3.

ALTERNATE BACKOFF ALGORITHM

In the case of IEEE 802.3, the Backoff time after a collision is as discussed above. The random number R determines the Backoff time. The range of random number in turn depends on the number of retries. For example, on first transmission attempt, retry variable is zero, hence R could be either 0 or 1, resulting in a Backoff time of Interframe Spacing or one slot time. On the first retry, retry variable is equal to 1, and the range for R is 0, 1, 2, 3, resulting in Backoff times of Interframe Spacing, or one, two or three times the slot time. In the first case, the probability of two stations backing off for the same period of time is 50% (IFS or 1 slot time). In the second case, this probability will reduce to 25% and so on.

In some applications it may be desirable to resolve the contention for the channel faster. If the colliding stations were to pick the backoff random number from a larger range of numbers, the probability of the stations picking the same random number (and hence colliding again) will be reduced. The 82586 offers this capability by programming the alternate backoff method. If this method is selected, the backoff period is computed as follows:

Backoff Time = R × Slot Time where

$0 \leq R < 2 \exp [\min (10, \text{Retry} + K)]$

Where K is the alternate backoff number, programmable between 0 and 7.

Note that for IEEE 802.3, K = 0.

3.0 82586 RECEIVE FUNCTIONS

This section describes how the 82586 processes received frames. The 82586 checks all frames that appear on the link, decides which frames should be passed to host memory, and checks them for errors.

3.1 Frame Reception

The 82586 recognizes the boundary of an incoming frame by monitoring the link. When the 82586 detects a carrier (data transitions) on the link and it is not transmitting or executing Action Commands (see section 8), it starts accepting the incoming bits. The frame Preamble provides bit synchronization and the frame SFD Field is used to locate the first bit of the Destination Address field, see Figure 3. The Preamble and SFD Field are discarded. The 82586 compares the incoming frame's Destination Address to the receiving station's Individual Address or Multicast Address; the Broadcast Address is also checked.

If there is an address match (and the frame satisfies the minimum frame size), the 82586 passes the Destination Address, Source Address, Length Field, and Information Field to system memory, calculates the CRC and verifies its correctness during reception of the FCS field. If there is not an address match, the 82586 never requests the system bus, and the 82586 becomes ready to receive the next frame.

If a frame is received in error (CRC violation, alignment, No Resources, DMA overrun), the 82586 automatically reclaims the memory used to store that frame. The next received frame is stored in the reclaimed memory.

When configured to End of Carrier delineation (IEEE 802.3), end of frame is indicated by the carrier going inactive. The number of bits after the SFD Field must be a multiple of eight, residual ('dribble') bits are discarded, and not included in the Frame Check Sequence. When there are residual bits the frame is 'misaligned.'

When configured to Bitstuffing delineation, the 82586 performs zero bit deletion, discards the EOF flag, and all following bits until end of carrier. Residual bits are discarded in a manner similar to the End of Carrier method. An error is reported if the carrier goes inactive prior to recognition of an EOF flag.

The minimum frame length is configurable in the range of 0 to 255 bytes. Any frame containing less than the minimum (configured) number of bytes is presumed to be a fragment resulting from a collision. A collision fragment, longer than 6 bytes, is either stored in memory and marked as a short frame, or its area is reclaimed (depending on the Save Bad Frame configuration).

3.2 Addressing

Addressing allows frames to be directed to one or more specific hosts. The 82586 provides flexible addressing techniques allowing a frame to be received by a single host, a group of hosts (multicast), or all hosts (broadcast). The chip checks the incoming Destination Address according to the three methods simultaneously, the frame is accepted if there is a match. A frame whose address does not match has no effect on the 82586 nor on any other component of the station.

The 82586 is normally configured with a specific Individual Address using the IA-SETUP command (see section 8.3). The default configuration is an all ones address 6 bytes long. An Individual Address is indicated by a zero in the least significant bit, and its length is determined by the configured Address Length parameter. During reception, the 82586 compares the incoming Destination Address with its Individual Address. All bits must be equal for an Individual Address match to be determined.

A frame may be targeted to all hosts (Broadcast) by using the Broadcast Address, an all ones address. During reception, the 82586 checks if the Destination Address is a Broadcast Address. If it is, an address match is confirmed. The 82586 can be configured to disable reception of frames with Broadcast Destination Addresses (for stations with limited storage resources).

The user can target a frame to a group of hosts. Using a method not involving the 82586, hosts are placed into groups, each group is assigned a Multicast Address. A host may belong to a number of groups and a group may contain a number of hosts. A host may address a frame to all hosts that belong to a particular group by specifying that group's Multicast-Address in the Destination Address field. A one in the least significant bit of the Destination Address distinguishes Multicast-Addresses from Individual Addresses.

The 82586 maintains a 64-bit Hash table. The 82586 maps every Multicast-Address into a single bit in the table. Using the MC-SETUP command (see section 8.5), the user provides a set of Multicast-Addresses, and the 82586 maps and stores them in the Hash table. During reception of a frame whose Destination Address is a Multicast-Address, the 82586 maps the Address, and checks if it appears in the Hash table. If it does, an address match is determined and the frame is passed to the host.

It is possible for more than one Multicast Address to be mapped into a given Hash bit. Thus, the host may have to perform additional checking. If 64 or fewer Multicast-Addresses are used in a system, it is possible to select address values that map into unique bits in the Hash table. The Hashing function is the CRC polynomial used for bit error detection. The 6 most significant bits (2–7), are selected from the first byte of the CRC shift register to index the 64-bit Hash table.

4.0 82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. Thus, it is particularly difficult to pinpoint a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 offers the four diagnostic services. First, monitoring the transmission and reception of frames. Second, statistics gathering and diagnostics of the Network as a whole. Third, diagnostic support for a particular station on the Network. Fourth, a means to test the proper operation of the chip itself.

4.1 Transmission/Reception Error Reporting

The 82586 stores, in system memory, status information after completing transmission or reception of every frame. If transmission or reception is successful, the OK status bit is set. If transmission is unsuccessful, the cause is given in the status. In case of unsuccessful reception, the cause is provided only if the 82586 is configured to Save Bad Frame, otherwise, only the statistics counters are updated.

The 82586 reports on the following events after each transmitted frame:

- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun occurred because the system bus did not keep up with the transmission.
- Transmission unsuccessful; the number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame. (CRC continues to be checked if the 82586 enters the NO RESOURCES state.)
- Alignment error: incorrect CRC in a misaligned frame. A misaligned frame with a correct CRC is not reported by the 82586.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- No EOF flag: valid only in Bitstuffing mode, carrier went inactive before EOF flag detection.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

4.2 Network Planning and Maintenance

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a

network management entity. The features include: gathering network activity information, updating error counters, saving all frames that appear on the link, and locating opens or shorts on the link.

Network Activity information is provided in the status returned to the host after each frame transmitted. The activity indicators are:

1) Number of collisions: the number of collisions the 82586 experienced in attempting to transmit this frame.

2) Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes address filtering, and is longer than the Minimum Frame Length configuration parameter. They reside in shared memory and are incremented by the 82586. The statistics registers provide the following information:

1) CRC errors: the number of frames that experienced a CRC error and were properly aligned.

2) Alignment errors: the number of frames that experienced a CRC error and were misaligned.

3) No-resources: the number of correct frames lost due to lack of memory resources.

4) Overrun errors: the number of frame sequences lost due to DMA overruns.

The 82586 can be configured to Promiscuous Mode. In this mode the 82586 captures all frames transmitted on the Network without checking the Destination Address. This mode is useful in implementing a monitoring station to capture all frames for network analysis.

Each 82586 is capable of determining if there is a short or open circuit anywhere in the network using the built in Time Domain Reflectometer (TDR) mechanism. When a TDR command (see section 8.7) is issued, the chip transmits a TDR frame and measures the reflection return time. If the network is properly terminated, there are no reflections so the timer runs out and the user is notified that there are no link problems. If a problem is detected, the distance to the reflection source and reason (short or open) are recorded.

4.3 Station Diagnostics

To support the testing of station hardware, the 82586 provides external loopback and Signal Quality Error test.

The 82586 can be configured to External Loopback. In this mode the 82586 operates full duplex at full speed. The maximum number of bytes in a frame to be looped back is 18 in the case of IEEE 802.3 (10 Mb/s) and 8 MHz system bus, but the actual maximum depends on bit rate and system bus speed. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Ethernet Serial Interface, the Transceiver cable, or in the Transceiver.

The IEEE 802.3 specification requires that the transceiver should verify the operation of the collision detect circuitry and pass the result to the controller. If the transceiver operates correctly, it sends a short 10 MHz pulse train to the controller on the Collision Detect pair. This mechanism is called the 'heart beat,' or Signal Quality Error Test, SQE TEST, signal. The SQE TEST signal is sent during the Interframe Spacing Time, just after completing transmission of a frame. The 82586 returns detection of the SQE TEST in status returned to the host at completion of transmission.

4.4 82586 Self Testing

The 82586 provides several features to check the chip operation.

The 82586 can be configured to Internal Loopback (section 11.5). In this mode, the 82586 disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock. Internally, the Transmit Clock is divided by 4 to allow internal full duplex operation. Internal Loopback overrides External Loopback. Equality between the transmitted and received frames implies that a large portion of the chip operates correctly. In addition, internal loopback can be used in conjunction with inhibiting the source address insertion and/or CRC insertion by the chip. For example: in internal loopback, if a frame is transmitted with an erroneous CRC (using CRC inhibition), the CRC checking mechanism must detect a CRC error.

The Dump Command (see section 8.8) causes the 82586 to write over 100 bytes of its internal registers to memory. This is a very powerful capability that can serve as the basis for comprehensive diagnostics.

There are parts of the chip, in particular the logic that uses the exponential Backoff random number generator that cannot be checked from the outside. The Diagnose Command (see section 8.9) initiates a self test procedure that exercises the otherwise unaccessible registers and counters, and reports the result.

5.0 82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's direct access to memory capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 operates easily with general purpose processors, however, a minimum hardware configuration is realizable with the 80186/80188 processors. In discussing 82586/Host interaction, the logical interface and the hardware bus interface are referred to separately.

5.1 Logical Interface

The 82586 consists of two independent units: the Command Unit (CU) and the Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The Shared Memory structure is composed of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA), see Figure 4.

The Initialization Root is at a predetermined location in the memory space, (0FFFFF6H), known to both the host CPU and the 82586. The root is accessed at initialization and points to the System Control Block. Section 6 provides details on the initialization root and the algorithms performed by the 82586 during initialization.

The System Control Block (SCB) serves as a bidirectional mailbox between the host CPU, CU and RU. It is the central element through which the CPU and the

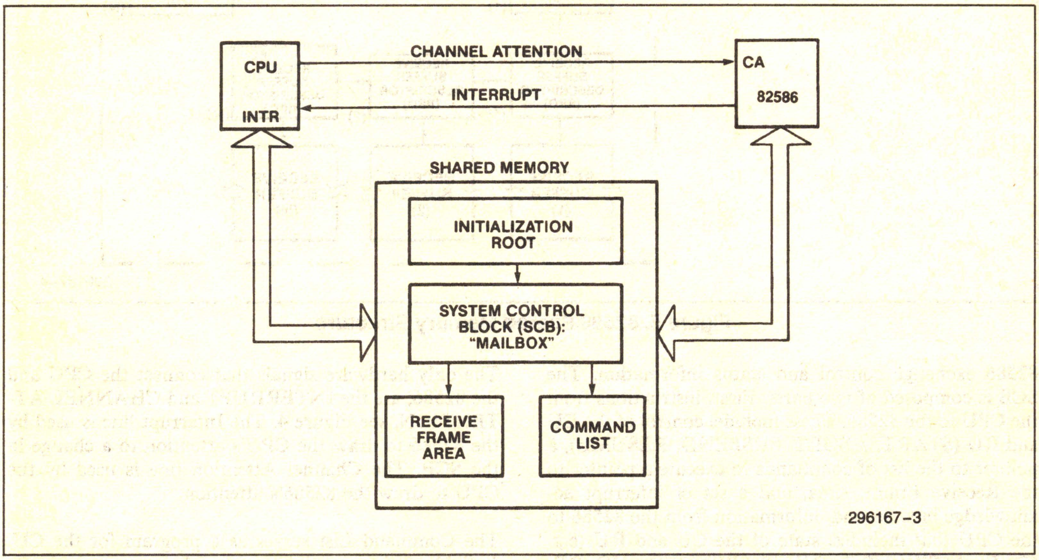


Figure 4. 82586/Host CPU Interaction

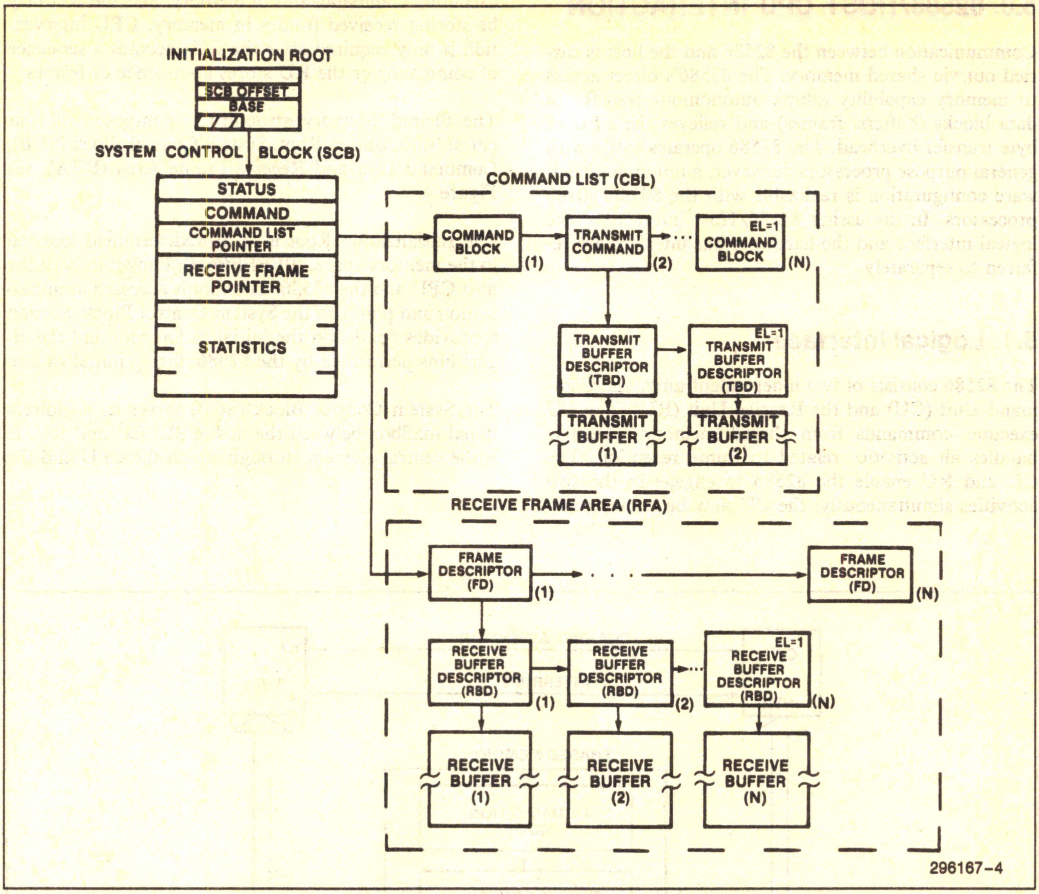


Figure 5. 82586 Shared Memory Structure

82586 exchange control and status information. The SCB is composed of two parts. First, instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands to execute a pointer to the Receive Frame Area, and a set of interrupt acknowledge bits. Second, information from the 82586 to the CPU that includes: state of the CU and RU (e.g. IDLE, ACTIVE/READY, SUSPENDED, NO RECEIVE RESOURCES), interrupt bits (command completed, frame received, CU gone not ready, RU gone not ready), and statistics. See Figure 5. Section 7 provides a detailed description of the SCB.

The only hardware signals that connect the CPU and the 82586, are the INTERRUPT and CHANNEL ATTENTION, see Figure 4. The Interrupt line is used by the 82586 to draw the CPU's attention to a change in the SCB. The Channel Attention line is used by the CPU to draw the 82586's attention.

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CBs contain command specific parameters and command specific statuses. Specifically, these commands are called Action Commands (e.g. Transmit, Configure) and are discussed in detail in section 8.

A specific command, Transmit, causes transmission of a frame by the 82586. The Command Block includes Destination Address, Length Field, and a pointer to a list of linked buffers that hold the frame to be constructed from several buffers scattered in memory. The Command Unit performs in parallel, without the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and list of buffers prepared by the user. It is conceptually distinct from the Command List. Because frames arrive without being solicited by the 82586, the 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in IEEE 802.3) for each frame. Maximum frame length buffers are inefficient because 75 percent of the frames on a network are control (request for status, message acknowledgement, etc.) frames, which are typically less than 100 bytes. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was CPU processing overhead and the time involved in buffer switching (especially at 10 Mbps). The 82586 overcomes this drawback by performing buffer management in hardware, completely transparent to the user.

Section 9 provides details on the format of the Received Frame Area and the algorithms associated with frame reception.

5.2 Hardware Bus Interface

The local bus interface has 24 address lines. The low order 16 lines are multiplexed with data, and address line number 19 is multiplexed with status. Like other Master peripherals, the 82586 provides all control signals required to handle Direct Memory Access. The bus interface operates at up to 8 MHz.

Similar to the 8086, the 82586 can be pin strapped into either Minimum mode or Maximum mode. Minimum mode is used in small systems that do not require external bus controller circuitry. The 82586 provides the minimum bus control information. Maximum mode is used in systems that use large memory, and have system peripherals.

The maximum Data Transfer Rate on the bus interface is 4 megabytes per second. Note that this is significantly higher than required by the IEEE 802.3 (which is 1.25 megabytes per second). This leaves much bus bandwidth for 82586 buffer, status and control overhead and general application processing. Although the 82586 performs command chaining, frame chaining, and buffer chaining on the fly, it can operate with buses that transfer data as slow as 2 megabytes per second without DMA overruns or underruns.

The 82586 shares the system bus with the host CPU, and possibly other peripherals. Therefore, there is a delay between the time the 82586 requests the bus and receives it. This delay is referred to as bus latency. Latency time is typically 1 to 2 microseconds but may reach about ten microseconds in worst case situations. Note that 10 microseconds is equivalent to 100 data bits, or 12.5 bytes. DMA overruns or underruns due to bus latency is significantly reduced by the 82586's individual on-chip transmit and receive FIFOs. Associated with the FIFOs is a user programmable threshold mechanism that improves the bus access efficiency by making the traffic bursty.

Section 10 provides details on all aspects of the Bus Interface.

5.3 Memory Addressing

The 82586 accesses memory with 24-bit addresses (in Minimum mode the two most significant bits are used as $\overline{RD}$ and $\overline{WR}$ lines). The memory structure uses two types of address representation: Physical (Real) and Segmented.

A Physical Address is a single 24-bit entity that specifies the physical byte address in memory. The representation of this type of address is in three consecutive bytes in memory, starting at an even location (see Figure 6). It is used for referring to all the buffers as well as to elements of the Initialization Root. When the chip is configured to operate with a 16-bit bus, the Physical Address must be even (least significant bit = 0).

A Segmented Address consists of a base and offset. The base is a 24 bits long real address and the offset is

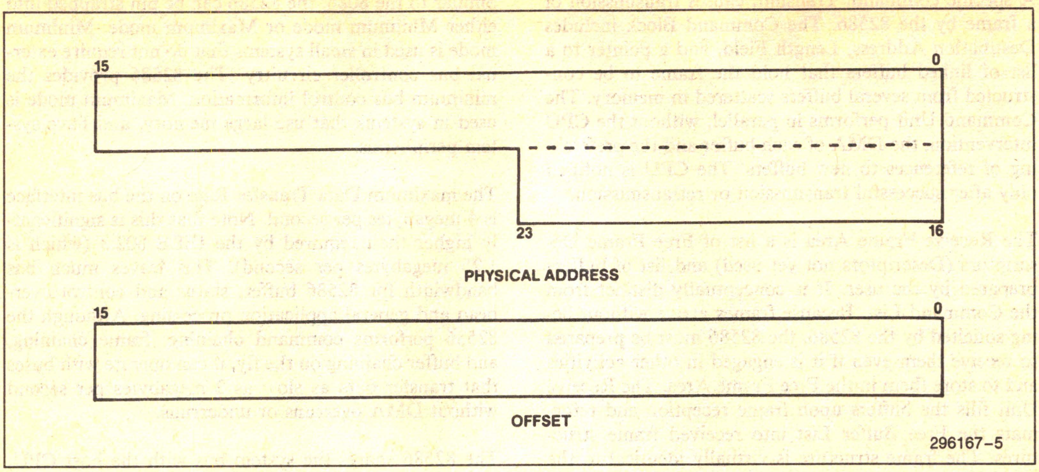


Figure 6. Memory Addressing

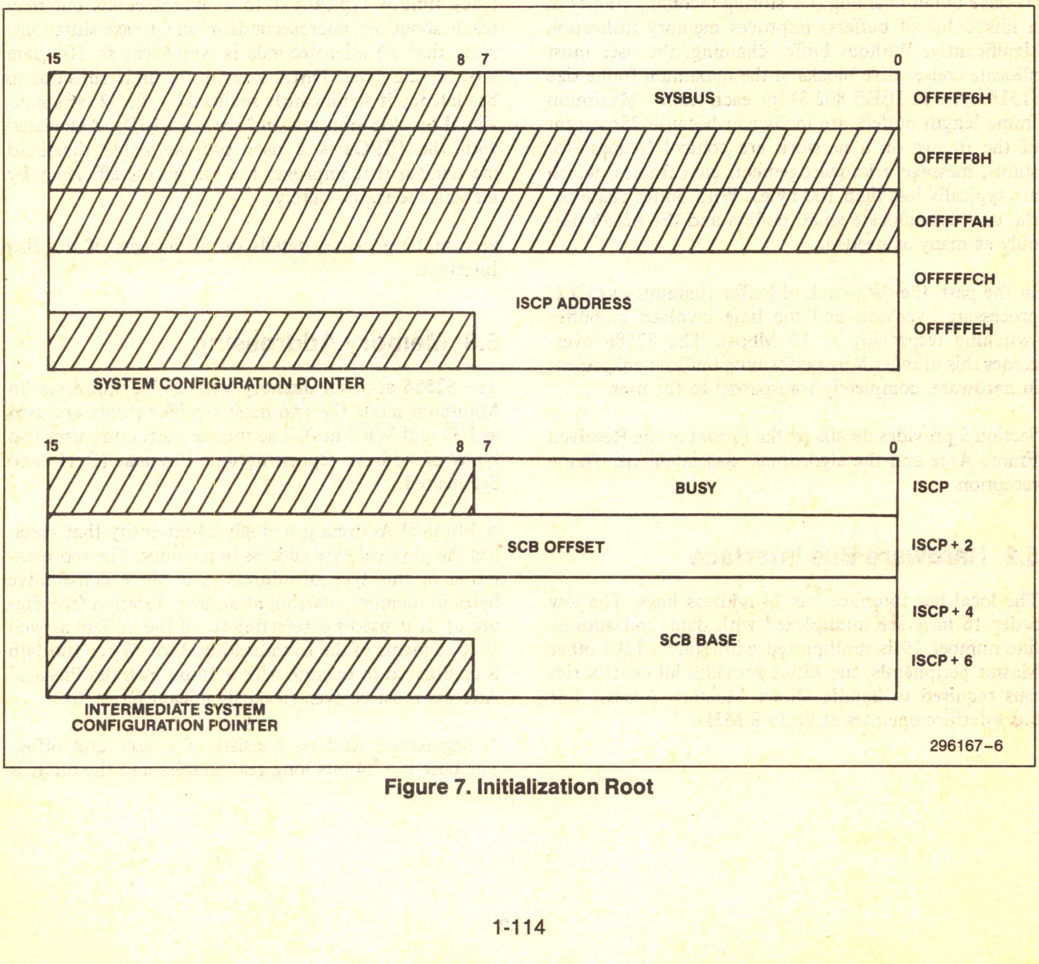


Figure 7. Initialization Root

16 bits long. The base is setup during the initialization process and remains the same until the chip is reset. The base must be even in all cases. The 82586 calculates the physical address by adding the base to the offset. The representation of the offset is a word starting on an even byte boundary in memory, (see Figure 6). Segmented Addresses are used for referring to most parts of the memory structure. It must be even in all cases.

6.0 INITIALIZING THE 82586

After the clocks (System, Transmit and Receive) applied to the 82586 have stabilized and a RESET is issued, the 82586 performs an initialization procedure that prepares it for normal operation. This section specifies the memory structure involved in the initialization, how the user must prepare it, and how the chip behaves during initialization.

6.1 Initialization Root Format

Initialization involves the System Configuration Pointer (SCP) and the Intermediate System Configuration Pointer (ISCP). Their formats are described in detail below, and they are shown in Figure 7.

SYSTEM CONFIGURATION POINTER (SCP)

The SCP is located in the ten bytes from 0FFFFF6H to 0FFFFFFH, the highest bytes in the address space. Note that this is a fixed address, the only one in the system.

Only two items of information are present in the SCP, the system bus width and a pointer to ISCP, the next structure. ISCP is an arbitrary location in the address space. The SCP includes the following fields:

0FFFFF6H:

BUS (Bit 0) - This bit specifies the width of the system bus. It takes the following values:

0 16-bit bus

1 8-bit bus

ISCP ADDRESS: This 24-bit value is the ISCP physical address.

The 82586 is indifferent to the remaining SCP contents. This does not mean that these areas have no function, they may be meaningful with respect to other peripherals which refer to SCP. Because the SCP and ISCP are sampled by the 82586 only during initialization, they may be used at any other time by other peripherals.

INTERMEDIATE SYSTEM CONFIGURATION POINTER (ISCP)

The ISCP is common to the 82586 and all other master peripherals referring to the same SCP. At the same time however, information contained in ISCP is specific to each master peripheral, and the structure contains a status element. Consequently, ISCP must be located in RAM, and not in ROM with SCP. This distinction is the reason for separating SCP and ISCP. The ISCP includes the following fields:

BUS Y (8 bits): when set to 01H, indicates that the 82586 is in the initialization process. The 82586 clears this byte immediately after reading the information contained in ISCP.

SCB-OFFSET: the address offset of the SCB (within the segment defined by the SCB-BASE).

SCB-BASE: a 24-bit value giving the starting address of the 64-kilobyte segment containing SCB and all other control structures dealing with the 82586.

6.2 Initialization Process

The initialization process establishes communication between the CPU and the 82586. Without it, no interaction takes place.

Prior to starting the initialization process, the CPU must setup the following fields in SCP and ISCP:

- BUS - memory bus width specification.
- ISCP ADDRESS - physical address of ISCP.
- BUSY - this field in ISCP must be 01H to indicate that the CPU is ready for initialization. Subsequent clearing of the field by the 82586, indicates initialization completion.
- SCB BASE - base address of the entire memory structure.
- SCB OFFSET - offset (from SCB BASE) of SCB. After completing initialization, all 82586 control is via SCB.

The CPU must reset the 82586. After power-up, this must be done with a hardware RESET. Subsequently, a software RESET (specific to the 82586) may be used. RESET causes all major internal flags to be set to their inactive states. In particular, CU and RU are set to IDLE state and configuration parameters are set to their default values (see section 7.5).

Initialization is triggered by the Channel Attention (CA) signal from the CPU. Note that initialization can only occur by issuing the sequence: RESET-CA. After CA, the 82586 performs the following sequence:

1) Reads the first word from location 0FFFFF6H. The least significant bit determines bus width of all subsequent memory accesses. Before reading this word, the system bus width is assumed to be 8 bits.

2) Reads the ISCP ADDRESS from location 0FFFFFCH and the following word.

3) Reads SCB OFFSET from the word following the one determined by ISCP ADDRESS. SCB OFFSET is saved for subsequent references to SCB.

4) Reads SCB BASE field from the next two words.

5) Clears the BUSY byte. This is done by reading the word in location ISCP, clearing the least significant byte, and writing back to the word (with the cleared byte).

6) The SCB BASE (read in Step 4) is saved internally to serve as a base for all subsequent references to the memory structure (excluding data buffers).

7) Writes the STATUS word of SCB with CX and CNA bits set, all remaining fields are cleared (see section 7.1).

8) Raises the INTERRUPT Hardware Signal.

The chip is now ready to be controlled by the host CPU via SCB, as described in the next section.

7.0 CONTROLLING THE 82586

This section discusses how the CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU). Operation of the CU and RU themselves, namely how the 82586 executes Action Commands and receives frames, is discussed in subsequent sections. This section provides complete explanations of the following functions:

- Starting and Completing Control Commands
- CU Control
- RU Control
- RESET
- Statistics Registers

7.1 System Control Block (SCB) Format

The System Control Block is the communication mailbox between the 82586 and the host CPU. The SCB format is shown in Figure 8.

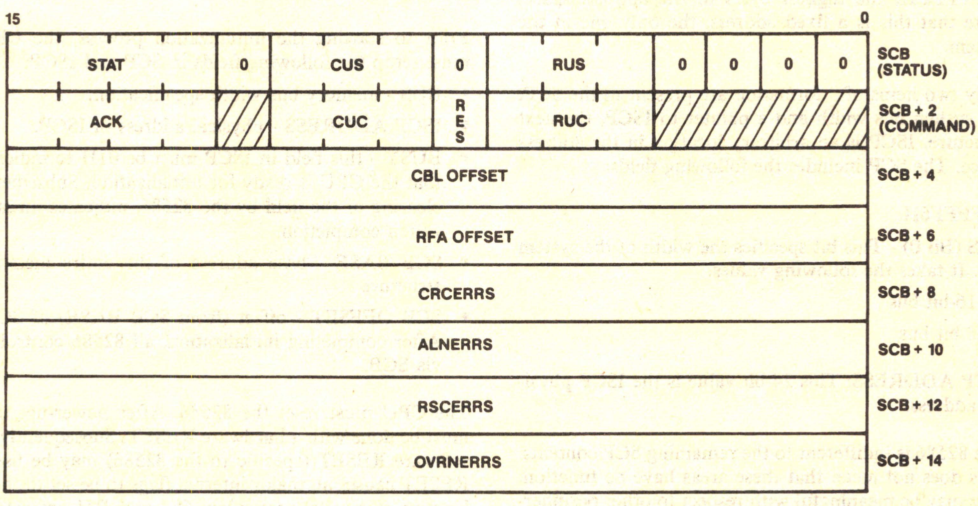


Figure 8. System Control Block (SCB) Format

The host CPU issues Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. The CPU informs the 82586 that a control command is ready by issuing a CA signal.

The SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

STATUS WORD:

Indicates status of the 82586 to the CPU. This word is modified only by the 82586. Defined bits are:

CX (Bit 15) - The CU finished executing an Action Command with its 'I' (Interrupt) bit set.

FR (Bit 14) - The RU finished receiving a frame.

CNA (Bit 13) - The CU left the ACTIVE state.

RNR (Bit 12) - The RU left the READY state.

CUS (Bits 8–10) - (3 bits) this field contains the status of the Command Unit. Valid values are:

- 0 - IDLE
- 1 - SUSPENDED
- 2 - ACTIVE
- 3–7 - Not used

RUS (Bits 4–6) - (3 bits) this field contains the status of the Receive Unit. Valid values are:

- 0 - IDLE
- 1 - SUSPENDED
- 2 - NO RESOURCES
- 3 - Not used
- 4 - READY
- 5–7 - Not used

The remaining bits of the status word are always set to zero by the 82586.

COMMAND WORD:

Provides the communication mechanism from the CPU to the 82586. This word is set by the CPU and cleared by the 82586. Defined bits are:

ACK-CX (Bit 15) - Acknowledges that the CU completed an Action Command.

ACK-FR (Bit 14) - Acknowledges that the RU received a frame.

ACK-CNA (Bit 13) - Acknowledges that the Command Unit left the ACTIVE state.

ACK-RNR (Bit 12) - Acknowledges that the Receive Unit left the READY state.

CUC (Bits 8–10) - (3 bits) this field contains the command to the Command Unit. Valid values are:

- 0 - NOP (doesn't affect current state of the unit).
- 1 - Start execution of the **first** command on the Command List (CBL). If a command is in execution, then complete it before starting the new CBL. The beginning of the CBL is in CBL OFFSET.
- 2 - Resume the operation of the Command Unit by executing the next command. This operation assumes that the Command Unit has been previously suspended.
- 3 - Suspend execution of commands on the CBL after current command is complete.
- 4 - Abort current command immediately.
- 5–7 - Reserved, illegal for use.

RUC (Bits 4–6) - (3 bits) This field contains the command to the receive unit. Valid values are:

- 0 - NOP (does not alter current state of unit).
- 1 - Start reception of frames. If a frame is being received, then complete reception before starting. The beginning of the RFA is contained in the RFA OFFSET.
- 2 - Resume frame receiving (only when in the SUSPEND state.)
- 3 - Suspend frame receiving. If a frame is being received, then complete its reception before suspending.
- 4 - Abort receiver operation immediately.
- 5–7 - Reserved, illegal for use.

RESET (Bit 7) - Reset chip (logically the same as hardware RESET).

CBL-OFFSET: Gives the 16-bit offset address of the first command (Action Command) in the Command List to be executed following CU-START. Thus, the 82586 will read this word only if the CUC field contains a CU-START Control Command.

RFA-OFFSET: Gives the 16-bit offset address of the first Receive Frame Descriptor in the Receive Frame Area, to be accessed following a RU-START Control command. Thus, the 82586 will read this word only if the RUC field contains a RU-START Control Command.

CRCERRS: CRC Errors - contains the number of properly aligned frames received with a CRC error.

ALNERRS: Alignment Errors - contains the number of misaligned frames received with a CRC error.

RSCERRS: Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or Receive Frame Descriptors).

OVRNERRS: Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available to the 82586 in time to transfer them.

7.2 Starting and Completing Control Commands

The CPU issues Control Commands by writing in the SCB COMMAND field and issuing Channel Attention (CA). Acceptance of a Control Command is indicated by the 82586 clearing the SCB COMMAND field. Therefore, the CPU must wait for this word to be cleared before the next Control Command can be issued.

The 82586 does not necessarily accept the Control Command immediately after CA is issued; it may be engaged in higher priority tasks. For example, prefetching new buffers, handling buffer switches, or finishing frame reception. When the 82586 becomes available it performs the Starting of Control Command sequence:

1) Clears hardware Interrupt signal.

2) Reads the SCB COMMAND word, and analyzes its fields.

3) If the RESET bit is set, it performs the RESET sequence, as described in section 7.5.

4) Clears the internal INTERRUPT-IS ('In-Service') flag for each acknowledged interrupt bit.

5) If the RUC field is different from zero, an internal request to the RU to perform a RU command acceptance sequence is issued (see section 7.4).

6) If the CUC field is different from zero, the acceptance sequence for a CU command is performed immediately (see section 7.3).

After both CU and RU complete the acceptance sequence, the 82586 performs the Completion of Control Command sequence:

1) If there is any internal interrupt request then it sets the respective INTERRUPT-IS flag.

2) Update SCB STATUS word, according to internal CU status, RU status and Interrupt requests.

3) If any INTERRUPT-IS is set then the 82586 sets the hardware Interrupt signal.

4) Clears SCB COMMAND word.

7.3 Command Unit (CU) Control

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

This section describes how the CPU controls Action Command execution, namely, how it starts, stops, suspends or resumes the CU. Execution of the Action Commands themselves, is the subject of the next section.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - the CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- SUSPENDED - the CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- ACTIVE - the CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command. In general, the CPU can cause the CU to do the following:

- Start executing a list of Action Commands.
- Suspend execution after completing the current Action Command.
- Resume execution if the CU is in the SUSPENDED state,
- Abort execution and return to the IDLE state.
- Stop execution after completing a particular Action Command. This will be the last CB in the list.
- Suspend execution after completing a particular Action Command.
- Have the 82586 issue Interrupts after completing particular Action Commands.

There are three points in time relevant to the execution of Action Commands:

- Acceptance time (of a control command)—the time following a Channel Attention issued by the CPU, the CU reads the Control Command and takes initial action.
- Beginning of Execution—the time the CU starts executing an Action Command. This may be following a Control Command or in continuing the list of Action Commands. The details are presented in section 8.1.
- Completion of Execution—the time the CU completes executing an Action Command. This is the decision time for the CU on how to proceed. The details are presented in section 8.1.

The CU uses 3 internal flags to remember requests from acceptance time to be acted upon at completion of execution: CU-START-REQUEST, CU-SUSPEND-REQUEST and CU-ABORT-REQUEST.

At acceptance Time, after the 82586 has finished higher priority tasks and a CA is detected, it reads the SCB command word and analyzes its fields. The higher priority tasks are: receive end of frame processing, receive or transmit buffer prefetching, retransmission due to collisions, completion of Action Commands. The CUC field is one of the following: CU-START, CU-SUSPEND, CU-RESUME or CU-ABORT.

CU-START

Starts execution of the first Action Command in the list. The CU performs the following sequence:

1) Reads CBL OFFSET and saves it as a pointer to the next CB to be executed.

2) If the CU is not in the ACTIVE state, it goes to the ACTIVE state and requests the beginning of the next CB.

3) If the CU is in the ACTIVE state, then it does nothing at Acceptance time. Beginning of the next CB (i.e. the new one) starts after completion time of the current CB.

CU-SUSPEND

Suspends operation of the CU after completing the current CB. The 82586 performs the following sequence:

1) If the CU is in the ACTIVE state, it sets the internal CU-SUSPEND-REQUEST flag. This flag causes CU to enter the SUSPENDED state at command completion time.

2) If the CU is not in the ACTIVE state, it ignores the CU-SUSPEND command.

CU-RESUME

Resumes CU operation. The 82586 performs the following:

1) If the CU is in the SUSPEND state, it goes to the ACTIVE state and requests the beginning of the next CB.

2) If the CU is in the ACTIVE state, it clears internal CU-SUSPEND-REQUEST to prevent suspension at command completion time.

3) If the CU is in the IDLE state, it ignores the CU-RESUME command.

CU-ABORT

Causes the CU to stop all activity immediately and go to the IDLE State. The CU performs the following:

1) If the CU is not in the ACTIVE state, it goes to the IDLE state.

2) If the CU is in the ACTIVE state, it sets the internal CU-ABORT-REQUEST flag. This flag causes the CU to go to the IDLE state at command completion time.

The CU ABORT control command causes the immediate termination (at acceptance time) of the transmission process. The effect on specific Action Commands (see section 8) is as follows:

- NOP, TDR, DUMP and DIAGNOSE commands are not stopped. Following execution, the CU switches to its IDLE state.

- IA setup, MC setup, CONFIGURE commands are always stopped. A command aborted bit is set in the CB status field. The CPU MUST execute these commands again since the 82586 may not be properly programmed to continue its operation.
- There is an attempt to abort a TRANSMIT. The DMA process is stopped. In certain timings, the Abort attempt may not be successful. The 82586 internally tests for this situation and flags CMD ABORTED status only if the TRANSMIT is known to have failed.

NOTES ON ABORTED TRANSMISSION:

1) The CU attempts to stop as soon as possible by stopping DMA and forcing a FIFO underrun.

2) Aborted transmit frames may result in small frames on the Serial Link. The 82586 sends to the Serial Link, the partial frame and appends to it four bytes of 'All Ones' (jam pattern).

3) Apart from aborted transmit frames and collided frames, the 82586 appends the jam pattern, also in the following cases:
   - Underrun on transmit buffers.
   - Lost Carrier Sense, if programmed to stop transmission when Carrier Sense is lost.

See Figure 9.

SUSPENDING THE CU

The START, SUSPEND, RESUME and ABORT commands can be given through the SCB command fields as described above. The Command Block offset pointer in the SCB points to the first executable Action Command (to be discussed in section 8). It is sufficient to state here that these Action Commands have a field for EL and S Bits (see section 8.1). The EL, End of List, bit indicates that the current Action Command is the last on the Command List. The S Bit indicates that the user desires the CU to enter into Suspended state after executing the current command. Tables 1 and 2 illustrate the state transitions for various conditions.

Thus, there are two mechanisms to suspend the CU: one through the Suspend Command in SCB, and the other through the S bit in the Action Command Block. The Suspend Command in SCB will suspend the CU after completing the execution of the current command on the Command List, without knowing exactly which command the 82586 was executing before execution. Suspension of CU through Action Command Block will result in suspension after a specific command is executed. In both cases, the CU can be activated by issuing a Resume command.

Table 1. CU Control Commands: Actions at Acceptance Time

Present State	Start		Resume		Suspend		Abort	
	Next State	Action	Next State	Action	Next State	Action	Next State	Action
Idle	Active	Set Up CB	Idle	None	Idle	None	Idle	None
Suspended	Active	Set Up CB	Active	Set Up CB	Suspended	None	Idle	None
Active	Active	None	Active	None	Active	Request Suspend	Idle	Abort DMA/CNA Interrupt

Table 2. CU Activities Performed at End of Execution

EL Bit	S Bit	Request	Next State	Action
0	0	None	Active	Set Up CB
0	0	Suspend	Suspended	CNA Interrupt
0	1	None	Suspended	CNA Interrupt
0	1	Suspend	Suspended	CNA Interrupt
1	0	None	Idle	CNA, CX Interrupts
1	0	Suspend	Idle	CNA, CX Interrupts
1	1	None	Idle	CNA, CX Interrupts
1	1	Suspend	Idle	CNA, CX Interrupts

NOTES:
1) After a CB with 'I' bit set is completed, CX interrupt is generated.
2) Since the transition ACTIVE to ACTIVE STATE via the START Command is smoothly performed, no interrupt, related to state transition, is generated and no action is required at the end of Action Command Execution.

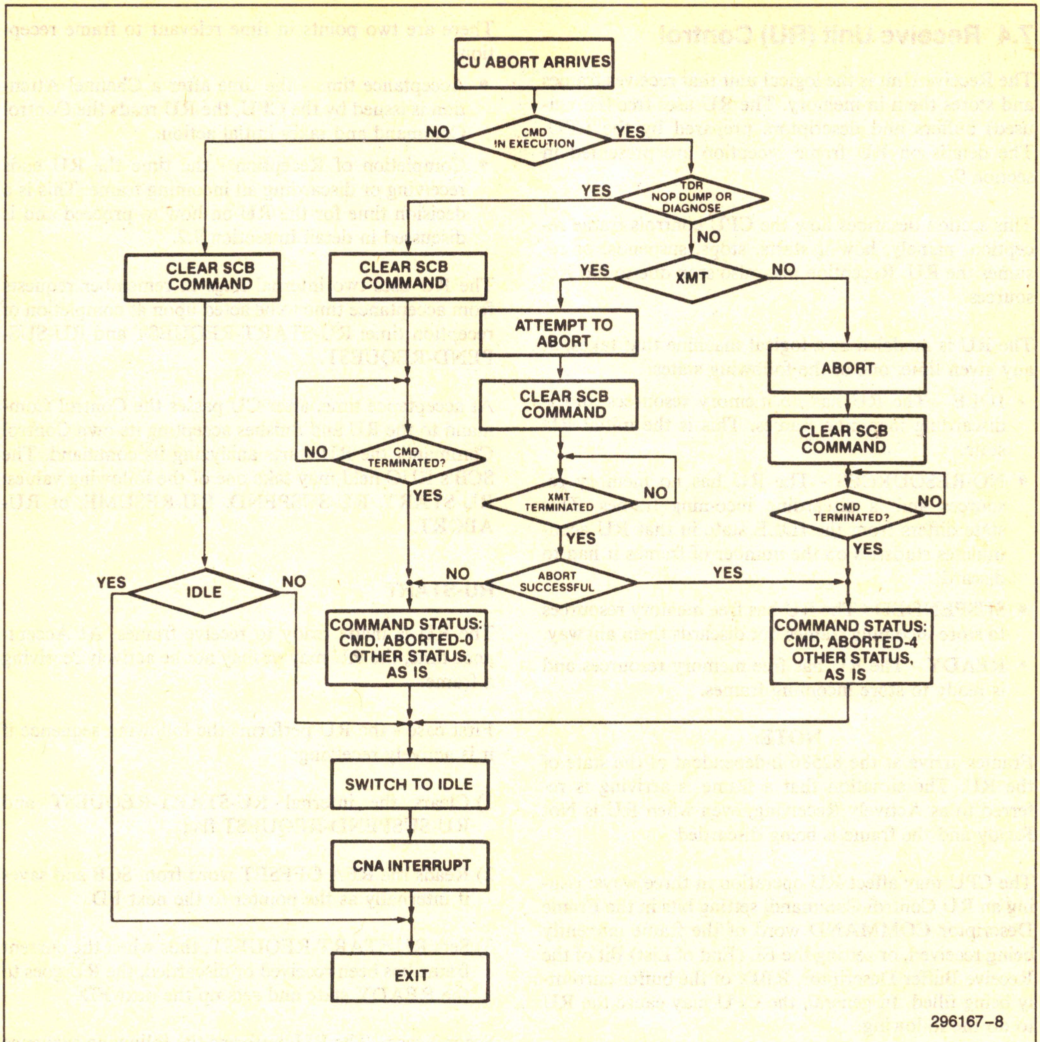


Figure 9. CU Abort Flowchart

7.4 Receive Unit (RU) Control

The Receive Unit is the logical unit that receives frames and stores them in memory. The RU uses free (i.e. unused) buffers and descriptors prepared by the CPU. The details on RU frame reception are presented in section 9.

This section describes how the CPU controls frame reception, namely, how it starts, stops, suspends, or resumes the RU. Reception may also stop due to No Resources.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - The RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- NO-RESOURCES - The RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- SUSPENDED - The RU has free memory resources to store incoming frames but discards them anyway.
- READY - The RU has free memory resources and is ready to store incoming frames.

NOTE:
Frames arrive at the 82586 independent of the state of the RU. The situation that a frame is arriving is referred to as Actively Receiving, even when RU is Not Ready and the frame is being discarded.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in the Frame Descriptor COMMAND word of the frame currently being received, or setting the EL (End of List) Bit of the Receive Buffer Descriptor, RBD, of the buffer currently being filled. In general, the CPU may cause the RU to do the following:

- Start frame reception.
- Suspend frame reception (start discarding) after current frame reception is complete.
- Resume reception if the RU is in SUSPENDED state.
- Abort reception at once and return to IDLE state.
- Stop reception after a particular FD is filled (frame received). This FD is referred to as the last FD on the list.
- Suspend reception after particular FD is filled (frame received).
- Stop reception after a particular RBD is filled. This is the last RBD in the list.

NOTE:
The RU issues an Interrupt after every received frame.

There are two points in time relevant to frame reception:

- Acceptance time - the time after a Channel Attention is issued by the CPU, the RU reads the Control Command and takes initial action.
- Completion of Reception - the time the RU ends receiving or discarding an incoming frame. This is a decision time for the RU on how to proceed and is discussed in detail in section 9.2.

The RU uses two internal flags to remember requests from acceptance time to be acted upon at completion of reception time: RU-START-REQUEST and RU-SUSPEND-REQUEST.

At acceptance time, after CU passes the Control Command to the RU and finishes accepting its own Control Command, the RU starts analyzing its command. The SCB's RUC field may take one of the following values: RU-START, RU-SUSPEND, RU-RESUME, or RU-ABORT.

RU-START

The RU is made ready to receive frames. At Acceptance time, the RU may or may not be actively receiving a frame.

First case - the RU performs the following sequence if it is actively receiving:

1) Clears the internal RU-START-REQUEST and RU-SUSPEND-REQUEST flag.
2) Reads the RFA OFFSET word from SCB and saves it internally as the pointer to the next FD.
3) Sets RU-START-REQUEST, thus when the current frame has been received or discarded, the RU goes to the READY state and sets up the next FD.

Second case - The RU performs the following sequence if it is not actively receiving:

1) Clears internal RU-START-REQUEST and RU-SUSPEND-REQUEST flags.
2) Reads RFA OFFSET word from SCB and saves it internally as the pointer to the next FD.
3) If the RU is not in the READY state, or the RU is in the READY state and assures that DMA did not transfer any data to the current FD, it does the following: it stops discarding, goes to the READY state, gives up buffers that have been prefetched (if any), and sets up a new FD. Setting up a new FD uses the pointer to the next FD to prepare reception of the next frame (see section 9).

4) If the RU is in the READY state and DMA started transferring data to memory before the RU had the opportunity to stop it, the following occurs: the RU goes to the NO-RESOURCES state (without issuing a RNR Interrupt) and sets the internal RU-START-REQUEST. Next, (in the particular case where RU does not manage to stop DMA in time) one frame is discarded and the user is notified by temporarily being in the NO-RESOURCES state.

The CPU must ensure that the 82586 is properly configured before starting the RU. This rule specifically applies to IA-SETUP, MC-SETUP, CONFIGURE Action Commands. Failure to perform CONFIGURE Action Command before stating the RU may result in unpredictable behavior of the receive process.

Failure to perform IA-SETUP before a RU START may result in a coincidental address match on a received frame.

RU-SUSPEND

Suspends RU operation after reception of current frame is complete, or reception of first frame to be received is complete. The RU performs the following:

1) If the RU is in the READY state, sets internal RU-SUSPEND request.

2) If the RU is not in the READY state, ignores the command.

RU-RESUME

Resumes frame reception. The RU performs the following sequence:

1) Clears RU-SUSPEND-REQUEST.

2) If the RU is in the SUSPENDED state, and not actively discarding a frame, it goes to the READY state and sets up a new FD (see section 9).

Table 3. RU Control Commands - Actions at Acceptable Time

		Start		Resume		Suspend		Abort	
	Present State	**Next State**	**Action**	**Next State**	**Action**	**Next State**	**Action**	**Next State**	**Action**
RU Not Actively Receiving	**Idle**	Ready	Set Up FD	Idle	None	Idle	None	Idle	None
	No Resources	Ready	Set Up FD	No Resources	None	No Resources	None	Idle	None
	Suspended	Ready	Set Up FD	Ready	Set Up FD	Suspended	None	Idle	None
		Ready	Set Up FD	Ready	None	Ready	Request Suspend	Idle	Start Discarding RNR Interrupt
RU Actively Receiving	**Idle**	Idle	Request Start	Idle	None	Idle	None	Idle	None
	No Resources	No Resources	Request Start	No Resources	None	No Resources	None	Idle	None
	Suspended	Suspended	Request Start	Suspended	Request Start	Suspended	None	Idle	None
	Ready	Ready	Request Start	Ready	None	Ready	Request Suspend	Idle	Abort DMA Start Discarding RNR Interrupt

Table 4. RU Activities Performed at End of Execution

EL Bit	S Bit	Request	Next State	Action
0	0	None	Ready	Set Up FD
0	0	Suspend	Suspended	RNR Interrupt
0	0	Start	Ready	Set Up FD
0	1	None	Suspended	RNR Interrupt
0	1	Suspend	Suspended	RNR Interrupt
0	1	Start	Suspended	RNR Interrupt
1	0	None	No Resources	RNR Interrupt
1	0	Suspend	No Resources	RNR Interrupt
1	0	Start	Ready	Set Up FD
1	1	None	No Resources	RNR Interrupt
1	1	Suspend	No Resources	RNR Interrupt
1	1	Start	Ready	Set Up FD

NOTE:
After a frame is received, FR interrupt is generated.

3) If the RU is in the SUSPENDED state and actively discarding a frame, it sets RU-START-REQUEST.

4) If the RU is not in the SUSPENDED state, it ignores the command.

RU-ABORT

RU stops reception immediately and goes to the IDLE state. RU performs the following:

1) If the RU is in the READY state, it requests an RNR Interrupt.

2) Stops all DMA activity and starts discarding incoming data.

3) Changes the RU to the IDLE state.

7.5 Reset

The 82586 has both a software and hardware RESET. After power up, a hardware RESET is required by the 82586. A Channel Attention following this reset will result in the 82586 accessing the System Control Pointer located at 0FFFFF6H. The first access is made on an 8-bit data boundary. Depending on the contents of location 0FFFFF6H, the subsequent access will be made on 8 or 16-bit data boundary.

A software RESET is available on the 82586 through bit 7 of the Control Command word in the System Control Block. It may be used after the 82586 has been initialized and it has the ISCP and SCP addresses. The software RESET is intended to operate selectively on a particular 82586.

The hardware RESET causes all major hardware control flip flops to be cleared. The CU and RU also clear their internal flags, they include:

END-OF-LAST-BUFFER = 0

ID-FULL = 0

RU-state = IDLE

CU-state = IDLE

CU-SUSPEND-REQUEST = 0

CU-ABORT-REQUEST = 0

CNA-REQUEST = 0

CX-REQUEST = 0

RU-SUSPEND-REQUEST = 0

RU-START-REQUEST = 0

RNR-REQUEST = 0

PR-REQUEST = 0

ADDRESS-LENGTH = 6

NOTE:
The System and Transmit clocks must be stable at their correct levels and timings before hardware RESET can be issued.

The CU performs the following upon recognition of a software RESET:

1) Terminates DMA activity.

2) Writes all zeros to SCB COMMAND word.

3) Triggers a hardware RESET.

NOTE:
A Channel Attention from the CPU must be delayed for at least 8 clocks after the SCB COMMAND word (with a RESET) is cleared. Several internal flip flops, including the internal Channel Attention flip flop, are cleared 8 clocks after the SCB COMMAND word is cleared therefore, a premature Channel Attention may not get recognized.

7.6 Error Statistics Registers

The last four words in the SCB are statistics registers that accumulate tallies on: CRC errors, Alignment errors, number of frames lost due to No Resources, number of frames lost due to DMA overruns.

The 82586 increments the registers at Completion of Reception time if the appropriate event happens and the register has not reached 0FFFFH. The registers are 'sticky,' i.e. they do not wrap around from 0FFFFH to zero. The registers can be cleared (or set to any value) only by the host CPU.

The CPU cannot update the register while the 82586 is incrementing it (restoring the old value plus one) because the 82586 performs the read counter/increment/write counter operation without relinquishing the bus. This is done to ensure that no logical contention exists between the 82586 and the CPU.

In a dual port memory configuration, the CPU should check the state of the counter immediately after updating it. This practice checks for the case where the 82586 placed into the counter the incremented 'old counter' value after the CPU has cleared the counter. Because the 82586 does not write to the counter when 0FFFFH is reached, the CPU may safely reset the counter without this check.

Incrementing the registers does not depend on the state of the Receive Unit. The CRC, Alignment and overrun error counters are incremented for each frame that experiences the specific error, was targeted for this specific station, and was longer than the Minimum Frame Length configuration parameter. The No Resources counter is incremented when a frame targeted to this station, has no error, and is lost or partially lost because there was no room in the memory structure.

The counters get incremented during External Loop Back operation, as well as in normal Transmit and Receive operation.

CRC ERROR COUNTER:

Contains the number of properly aligned frames received with a CRC error. It is incremented by the 82586 for every frame with a CRC error that was targeted for this specific station and was longer than the Minimum Frame Length.

ALIGNMENT ERROR COUNTER:

Counts the number of misaligned frames received with a CRC error. It is incremented by the 82586 for every misaligned frame with a CRC error that was targeted for this specific station and was longer than the Minimum Frame Length.

NO RESOURCES ERROR COUNTER:

Records the number of correctly received frames discarded due to the lack of memory resources (buffer space or received frame descriptors). It is incremented by the 82586 for every frame discarded due to lack of memory resources that was targeted for this specific station and did not experience any error.

OVERRUN ERROR COUNTER:

Counts the number of received frame sequences lost because the memory bus was not avaialble in time to transfer them. Note that more than one frame may get lost due to overrun, and in this case the counter is incremented only once.

7.7 SCB Status Update

The 82586 updates the SCB status word in the following events:

1) At the end of the initialization procedure, the 82586 writes SCB status denoting CU IDLE, RU IDLE, CX interrupt and CNA interrupt.

2) When a control command is accepted, the 82586 updates the SCB status and clears the SCB command word to denote 'acceptance completed.'

3) After execution of an Action Command. However, the CX bit setting is directly connected to the I bit.

4) After receiving a frame.

8.0 ACTION COMMANDS

The 82586 executes a 'program' that is made up of Action Commands in the Command List, CBL. As shown in Figure 10, each command contains the command field, status and control fields, link to the next Action Command in the CBL, and any command-specific parameters. This command format is called the Command Block.

The 82586 has a repertoire of 8 commands:

NOP	Transmit
Individual Address Setup	TDR
Configure	Diagnose
Multicast Address Setup	Dump

NOP:

This command results in no action by the 82586 other than the normal command processing, such as fetching the command and decoding the command field.

INDIVIDUAL ADDRESS SETUP:

This command is used to load the 82586's unique address. The unique address is contained in the parameter field of the command.

CONFIGURE:

The Configure command is used to load the 82586 with its operating parameters. Upon reset, the 82586 initializes to the IEEE 802.3 based parameters. If the user wishes to use any other values, the Configure command is used.

MULTICAST ADDRESS SETUP:

This command allows the programmer to setup one or more multicast addresses into the 82586. The multicast addresses to be setup are located in the parameter field of the command.

TRANSMIT:

One Transmit command is used to send a single frame. If more than one frame is to be sent, the host CPU can link multiple Transmit commands together. The destination address, Length field and pointer to buffers containing the data field are contained in the parameter field of the Transmit command (AL-LOC = 0).

TDR:

This command performs the Time Domain Reflectometry test on the coaxial cable. The TDR command is used to detect and locate cable faults caused by either short or open circuits on the coaxial cable.

DIAGNOSE:

The Diagnose command puts the 82586 through a self-test procedure and reports on the success or failure of the internal test.

DUMP:

This command causes the 82586 to dump its internal registers into memory. The registers included are those loaded by the Configure and Address Set-Up commands, plus status and other internal working registers.

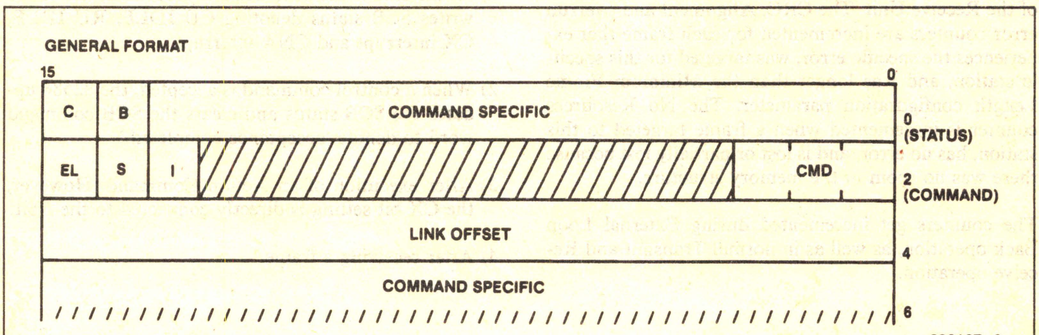


Figure 10. General Action Command

Table 5. 82586 Action Command Summary

- NOP, Code = 0
  - Parameters : None
  - Action : None
- IA-SETUP, Code = 1
  - Parameters : Individual Address
  - Action : Setup Individual Address
- CONFIGURE, Code = 2
  - Parameters : Byte count and configuration values list
  - Action : Configure the 82586
- MC-SETUP, Code = 3
  - Parameters : Byte count and Multicast Address list
  - Action : Setup Multicast HASH table
- TRANSMIT, Code = 4
  - Parameters : Destination Address, Length Field, linked buffer list holding the Data Field
  - Action : Transmit frame, and in case of collisions, retransmit
- TDR, Code = 5
  - Parameters : None
  - Action : Perform Time Domain Reflectometry test and return results
- DUMP, Code = 6
  - Parameters : Address of memory location to write results
  - Action : Dump set of internal registers to memory
- DIAGNOSE, Code = 7
  - Parameters : None
  - Action : Perform internal test procedure and returns results

8.1 General Action Command

The format common to all Action Commands and the algorithms for beginning and completing the execution (also common to all the commands) is described below.

NOTE:
System and Transmit Clocks must be applied to the 82586 before Action Commands can be executed.

GENERAL FORMAT

The general format of the Command Block (CB) includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- is always set at the completion of execution.
B	(Bit 14)	- is always set at the beginning of execution, and reset at completion of execution concurrently with C bit being set.

NOTE:
The 82586 does not read the B or C bits. If the SCB pointer points to a CB and C = 1, the 82586 will still execute the command.

COMMAND word:

EL	(Bit 15)	- Indicates that this is the last Command Block in the command list.
S	(Bit 14)	- Indicates that the CU should suspend after completion of execution of this command.
I	(Bit 13)	- Indicates that the 82586 should issue an interrupt after completion of executing this command.
CMD	(Bits 0–2)	- Indicates the specific command.

LINK OFFSET: Address of the next command block in the list.

BEGINNING OF EXECUTION

An Action Command may be started by either the CU-START or CU-RESUME Control Command, or may follow a previous Action Command. However, the actual command start may be delayed by RU activity.

The following sequence is performed by the CU at the beginning of execution of each Action Command:

1) Writes a word whose B bit is set and the remainder of the field cleared to the STATUS word of the next CB. The content of the STATUS word is specified for each Action Command in the description that follows.

2) The next CB becomes the current CB.

3) Reads the COMMAND word of the current CB and saves the following fields for later use: EL, I, S and CMD.

4) Reads the LINK OFFSET of the current CB and makes it the next CB.

5) Performs specific actions according to the Action Command specified in the CMD field.

COMPLETION OF EXECUTION

Command completion time is asynchronous to the beginning of the command. It is determined by the command type, RU activity, CU Control Commands, etc. The CU is always in the ACTIVE state at this time. The decision on how to proceed, depends on the following flags: CU-SUSPEND-REQUEST, CU-ABORT-REQUEST EL, I, S.

The following sequence is performed by the CU at completion of execution of an Action Command:

1) Writes command specific status to the STATUS word of the current CB.

2) If I bit is set, sets request for the CX Interrupt.

3) If EL or CU-ABORT-REQUEST is set, the CU becomes IDLE and generates request for CNA interrupt. If S or CU-SUSPEND-REQUEST is set, CU becomes SUSPENDED. Otherwise, requests beginning of next Action Command.

4) Sets appropriate INTERRUPT-IS flags and clears all Interrupt request flags.

5) Updates STATUS word in SCB.

6) Sets hardware Interrupt signal if any INTERRUPT-IS flag is set. The detailed description of commands is given below.

8.2 NOP

This command results in no action by the 82586, except as performed in normal command processing (section 8.1). It is present to aid in CBL manipulation.

NOP COMMAND FORMAT

NOP command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- NOP = 0

LINK OFFSET: Address of next Command Block

DETAILED OPERATION OF NOP COMMAND

CU performs the following sequence:

1) Starts Action Command (see section 8.1).

2) Prepares STATUS word with C = 1, B = 0, OK = 1.

3) Completes Action Command (see section 8.1).

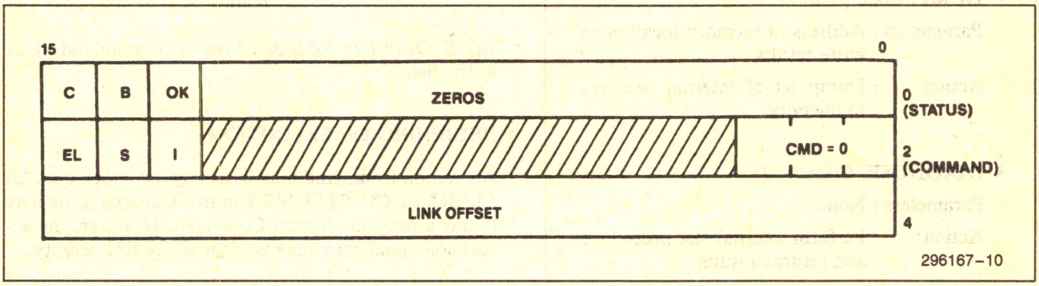


Figure 11. The NOP Command Block

8.3 IA-Setup

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

IA-SETUP COMMAND FORMAT

The IA-SETUP command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- IA-SETUP = 1

LINK OFFSET: Address of next Command Block

INDIVIDUAL ADDRESS: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

The 82586 will read only the number of bytes configured as address length.

DETAILED OPERATION OF THE IA-SETUP COMMAND

The Individual Address is transferred by TX-DMA via TX-FIFO to Transmit-Byte-Machine. See section 13. The CU performs the following sequence:

1) Starts Action Command.
2) Writes the IA-SETUP command byte to TX-FIFO.
3) Initiates TX-DMA with the address of the first byte of INDIVIDUAL ADDRESS and byte count according to configured Address Length.
4) Upon completion of DMA, writes the End of Command byte to TX-FIFO.
5) Waits for the Transmit-Byte-Machine to complete the updating of Individual Address register.
6) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
7) Completes the Action Command.

The Transmit-Byte-Machine maintains a 48-bit Individual Address register used for Source Address insertion during transmission and for Destination Address recognition during reception. RESET causes the Individual Address register to be set to all ones.

After the Transmit-Byte-Machine reads an IA-SETUP command from TX-FIFO, it reads the following bytes into the Individual Address register. The expected number of bytes is determined by the Address length configuration parameter. If Address length is less than 6 bytes then only part of the register is used. If fewer bytes than Address length are read from TX-FIFO (in case of abortion) only that number of bytes is updated.

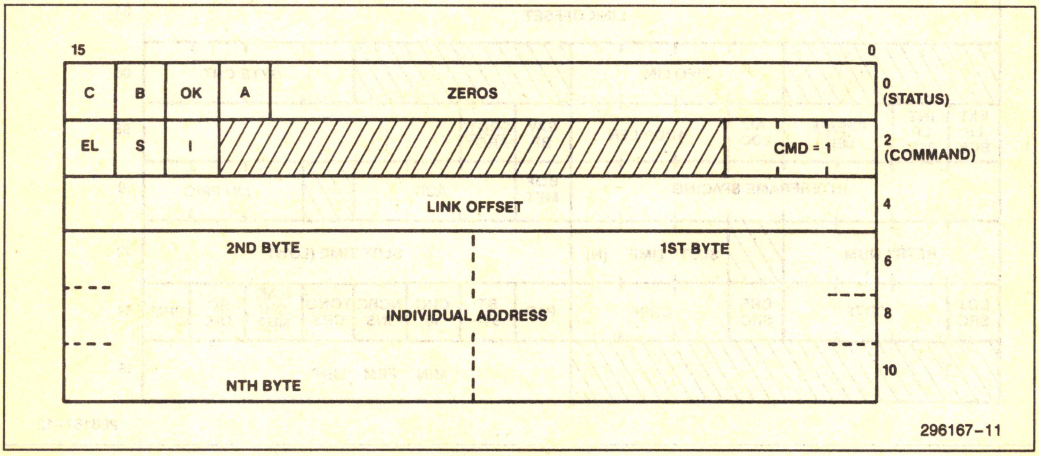


Figure 12. The IA-SETUP Command Block

8.4 Configure

The CONFIGURE command is used to update the 82586 operating parameters. It allows from 4 to 12 parameter bytes to be updated. Refer to section 12 for details on the configuration parameters.

CONFIGURE COMMAND FORMAT

The CONFIGURE command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- Configure = 2

LINK OFFSET: Address of next Command Block

Byte 6-7:

BYTE CNT (Bits 0–3) - Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12.

FIFO-LIM (Bits 8–11) - Value of TX-FIFO-Threshold

Byte 8-9:

SRDY/$\overline{\text{ARDY}}$ (Bit 6) - 0 - SRDY/ARDY pin operates as ARDY (internal synchronization).

1 - SRDY/ARDY pin operates as SRDY (external synchronization).

SAV-BF (Bit 7) - 0 - Received bad frames are not saved in memory.

1 - Received bad frames are saved in memory.

ADDR-LEN (Bits 8–10) - Number of address bytes. NOTE: 7 is interpreted as 0.

AL-LOC (Bit 11) - 0 - Address and Length Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted Frame, Source Address is inserted by the 82586.

1 - Address and Length Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).

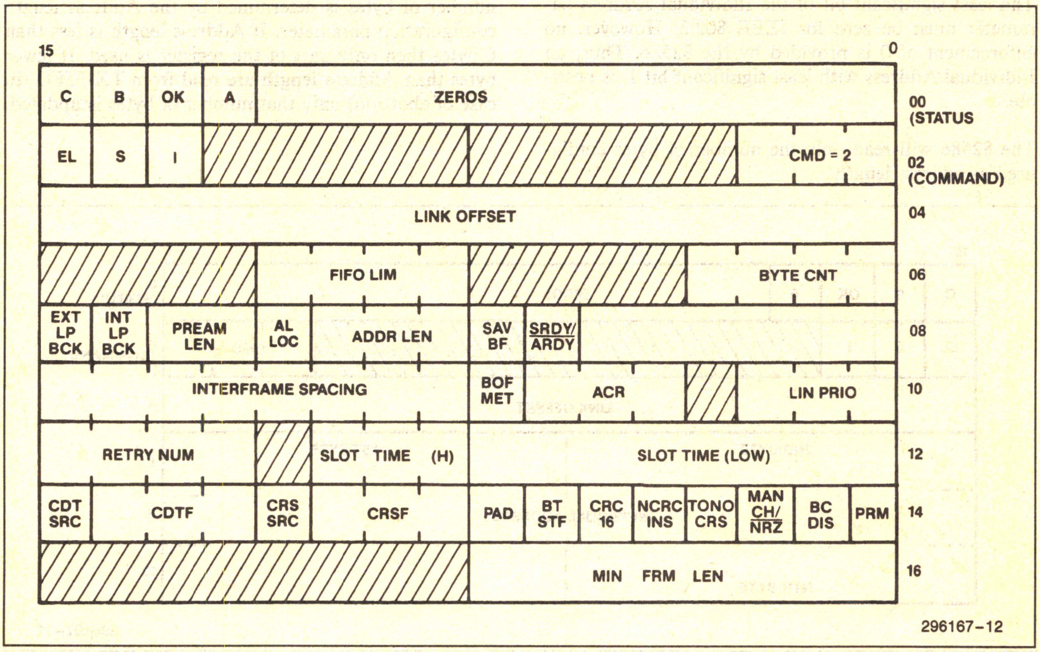


Figure 13. The CONFIGURE Command Block

PREAM-LEN (Bits 12–13) - Preamble Length including Start Frame Delimiter

00 - 2 bytes

01 - 4 bytes

10 - 8 bytes

11 - 16 bytes

INT-LPBCK (Bit 14) - Internal Loopback

EXT-LPBCK (Bit 15) - External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.

Byte 10–11:

LIN-PRIO (Bits 0–2) - Linear Priority

ACR (Bits 4–6) - Accelerated Contention Resolution

BOF-MET (Bit 7) - Exponential Backoff Method

0 - IEEE 802.3

1 - Alternate method

INTERFRAME-SPACING (Bits 8–15) - Number indicating the Interframe Spacing in TxC period units

Byte 12–13:

SLOT-TIME (L) (Bits 0–7) - Slot Time number, low byte

SLOT-TIME (H) (Bits 8–10) - Slot Time number, high bits

RETRY-NUM (Bits 12–15) - Maximum number of transmission retries on collisions

Byte 14–15:

PRM (Bit 0) - Promiscuous Mode

BC-DIS (Bit 1) - Broadcast Disable

MANCH/$\overline{\text{NRZ}}$ (Bit 2) - Manchester or NRZ encoding/decoding

0 - NRZ

1 - Manchester

TONO-CRS (Bit 3) - Transmit on No Carrier Sense

0 - Cease transmission if CRS goes inactive during frame transmission

1 - Continue transmission even if no Carrier Sense

NCRC-INS (Bit 4) - No CRC Insertion

CRC-16 (Bit 5) - CRC Type

0 - 32 bit Autodin II CRC polynomial

1 - 16 bit CCITT CRC polynomial

BT-STF (Bit 6) Bitstuffing:

0 - End of Carrier mode (IEEE 802.3)

1 - HDLC like Bitstuffing mode

PAD (Bit 7) - Padding

0 - No Padding

1 - Perform padding by transmitting flags for remainder of Slot Time

CRSF (Bits 8–9) - Carrier Sense Filter in bit times

CRS-SRC (Bit 11) Carrier Sense Source

0 - External

1 - Internal

CDTF (Bits 12–14) - Collision Detect Filter in bit times

CDT-SRC (Bit 15) - Collision Detect Source

0 - External

1 - Internal

Byte 16:

MIN-FRM-LEN (Bits 0–7) - Minimum number of bytes in a frame

CONFIGURATION DEFAULTS

The default values of the configuration parameters are compatible with the IEEE 802.3 Standard. RESET configures the 82586 according to the defaults shown in Table 6:

Table 6. 82586 Default Values

Preamble Length	=	2
Address Length	=	6
Broadcast Disable	=	0
CRC-16/CRC-32	=	0
No CRC Insertion	=	0
Bitstuffing/EOC	=	0
Padding	=	0
Min-Frame-Length	=	64
Interframe Spacing	=	96
Slot Time	=	512
Number of Retries	=	15
Linear Priority	=	0
Accelerated Contention Resolution	=	0
Exponential Backoff Method	=	0
Manchester/$\overline{\text{NRZ}}$	=	0
Internal CRS	=	0
CRS Filter	=	0
Internal CDT	=	0
CDT Filter	=	0
Transmit On No CRS	=	0
FIFO-Threshold	=	8
SRDY/$\overline{\text{ARDY}}$	=	0
Save Bad Frame	=	0
Address/Length Location	=	0
Internal Loopback	=	0
External Loopback	=	0
Promiscuous Mode	=	0

DETAILED OPERATION OF THE CONFIGURE COMMAND

Some parameters are kept by the CU, the remainder are transferred by TX-DMA to the Transmit-Byte-Machine via TX-FIFO. The CU performs the following sequence.

1) Starts the Action Command.
2) Reads bytes 6, 7, 8, and 9, and saves the following: BYTE-CNT, FIFO-LIM, SRDY/$\overline{\text{ARDY}}$, SAV-BF, ADDR-LEN, AL-LOC.
3) Writes the Configure byte to TX-FIFO.
4) Initiates TX-DMA to address of byte 6 and byte count specified by BYTE-CNT.
5) Upon completion of DMA writes the End of Command byte to TX-FIFO.
6) Waits for the Transmit-Byte-Machine to complete updating configuration registers.
7) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
8) Completes the Action Command.

The Transmit-Byte-Machine maintains 10 bytes of Configuration registers for determining the 82586's 'personality.' RESET causes Configuration registers to be set to values specified in Table 6 (compatible to IEEE 802.3).

After the Transmit-Byte-Machine reads the CONFIGURE command from TX-FIFO it reads and ignores 2 bytes, reads and updates Configuration registers with the next 10 bytes, and reads and ignores the remaining bytes. If less than 12 bytes are read when End of Command is encountered, only part of the Configuration registers are updated. The Transmit-Byte-Machine notifies CU after completion.

8.5 MC-Setup

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

MC-SETUP COMMAND FORMAT

The MC-SETUP command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B.	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- MC-SETUP = 3

LINK OFFSET: Address of next Command Block

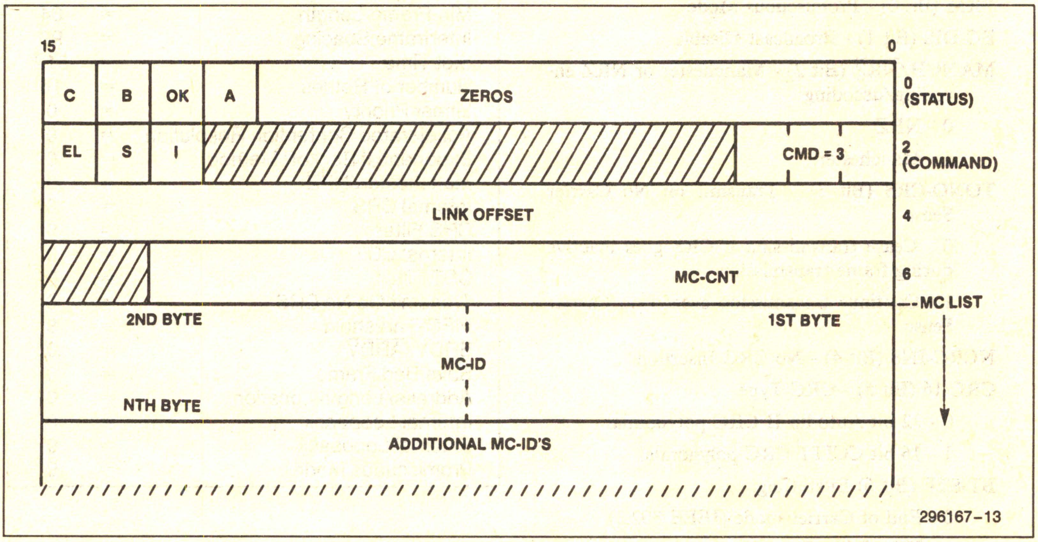


Figure 14. The MC-SETUP Command Block

MC-CNT: A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT = 0 disables reception of any incoming frame with a Multicast Address. The 14 bit field for MC addresses accommodates a maximum of 2730 Multicast Addresses, assuming a 6 byte address length.

MC-LIST: A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

DETAILED OPERATION OF MC-SETUP COMMAND

TX-DMA transfers the list of Multicast Addresses from memory to Transmit-Byte-Machine via TX-FIFO. See section 13. CU performs the following sequence:

1) Starts Action Command.
2) Reads MC-CNT and saves it internally.
3) Writes a MC-SETUP command byte to TX-FIFO.
4) Initiates TX-DMA with the MC-LIST address and byte count according to MC-CNT.
5) Upon completion of DMA, writes End of Command bit to TX-FIFO.
6) Waits for Transmit-Byte-Machine to complete HASH table update.
7) If the internal CU-ABORT-REQUEST flag is set, prepares STATUS word with C = 1, B = 0, OK = 0, A = 1; otherwise, prepares STATUS word with C = 1, B = 0, OK = 1, A = 0.
8) Completes the Action Command.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the Transmit CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2–7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

8.6 Transmit

The TRANSMIT command causes transmission and if necessary retransmission of a frame.

TRANSMIT COMMAND BLOCK FORMAT

TRANSMIT CB includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted
S10	(Bit 10)	- No Carrier Sense signal during transmission (between beginning of Destination Address and end of Frame Check Sequence). On the B-step this bit will always be zero when the 82586 is configured with TONO-CRS = 1. All subsequent steppings S10 will be set if the Carrier Sense signal is lost, regardless of the TONO-CRS Configure bit.
S9	(Bit 9)	- Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal.
S8	(Bit 8)	- Transmission unsuccessful (stopped) due to DMA underrun, (i.e. data not supplied from the system for transmission).
S7	(Bit 7)	- Transmission had to Defer to traffic on the link.
S6	(Bit 6)	- SQE TEST. Indicates that after the previous transmission and during the Interframe Spacing period, the SQE TEST signal was detected on the Collision Detect pin. This bit is meaningful if S5 and MAX-COLL fields are both zero, i.e. no collisions occurred during transmission.
S5	(Bit 5)	- Transmission attempt stopped due to number of collisions exceeding the maximum number of retries.
MAX-COLL	(Bits 3-0)	- Number of Collisions experienced by this frame. S5 = 1 and MAX-COLL = 0 indicates that there were 16 collisions.

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- TRANSMIT = 4

LINK OFFSET: Address of next Command Block

TBD OFFSET: Address of the first Transmit Buffer Descriptor. TBD-OFFSET = 0FFFFH indicates that there is no Data field.

DESTINATION ADDRESS: Destination Address of the frame.

LENGTH FIELD: Length Field of the frame.

TRANSMIT BUFFER DESCRIPTOR FORMAT

STATUS word:

EOF - Indicates that this is the Buffer Descriptor of the last buffer of this frame's Data Field.

ACT-COUNT (Bits 0–13) - Actual number of data bytes in the buffer (can be even or odd).

NEXT BD OFFSET: Points to next Buffer Descriptor on the list. When the EOF bit is set, this field is meaningless.

BUFFER ADDRESS: 24-bit absolute address of buffer.

DETAILED OPERATION OF TRANSMIT COMMAND

Execution of TRANSMIT Command causes transmission of a frame. If the frame experiences collisions, the CU retransmits the frame up to a configurable maximum number of times.

To transmit a frame, the user must setup the following:

- Command code.
- Destination Address and Length Field in TRANSMIT CB.
- TBD OFFSET should point to the first Buffer Descriptor of the list of buffers that holds the Data Field. If the frame does not include Data, TBD OFFSET should be set to 0FFFFH.
- For each buffer, BUFFER ADDRESS, ACTUAL COUNT and NEXT BD OFFSET should be setup. The EOF flag indicates the last buffer in the list representing the Data Field.

Unlike other Action Commands having parameters in one block in memory, the TRANSMIT Command may have parts of the parameters scattered in a linked list of buffers. The CU prefetches the buffers in the list on the fly. If the AL-Location configuration is zero, the Destination Address and Length Field are in the TRANSMIT CB and are treated as the first buffer. The Information Field is in the list of buffers. If the AL-Location Bit is one, the whole frame is in the buffers. See Figure 17.

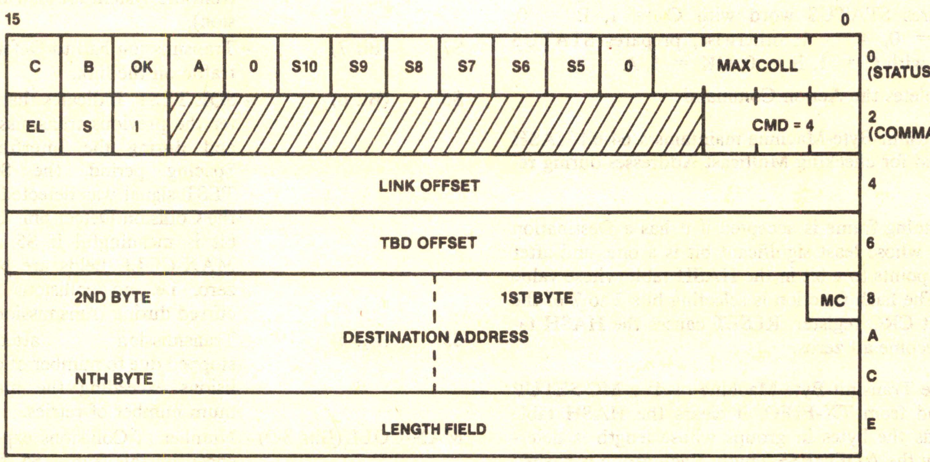


Figure 15. The Transmit Command Block

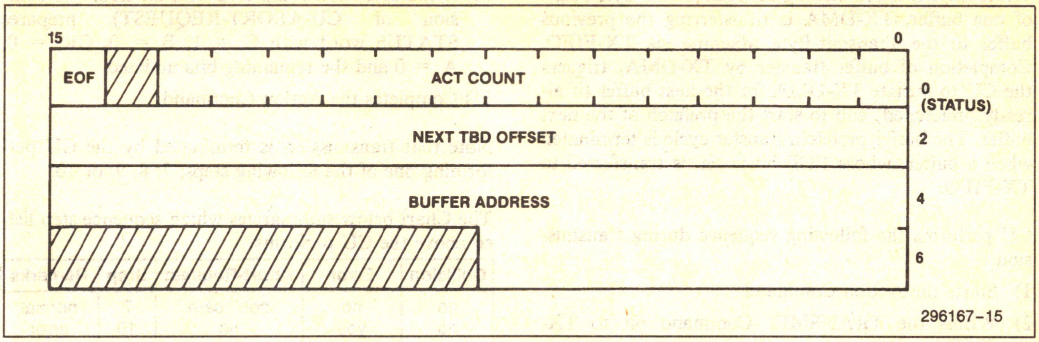


Figure 16. The Transmit Buffer Descriptor

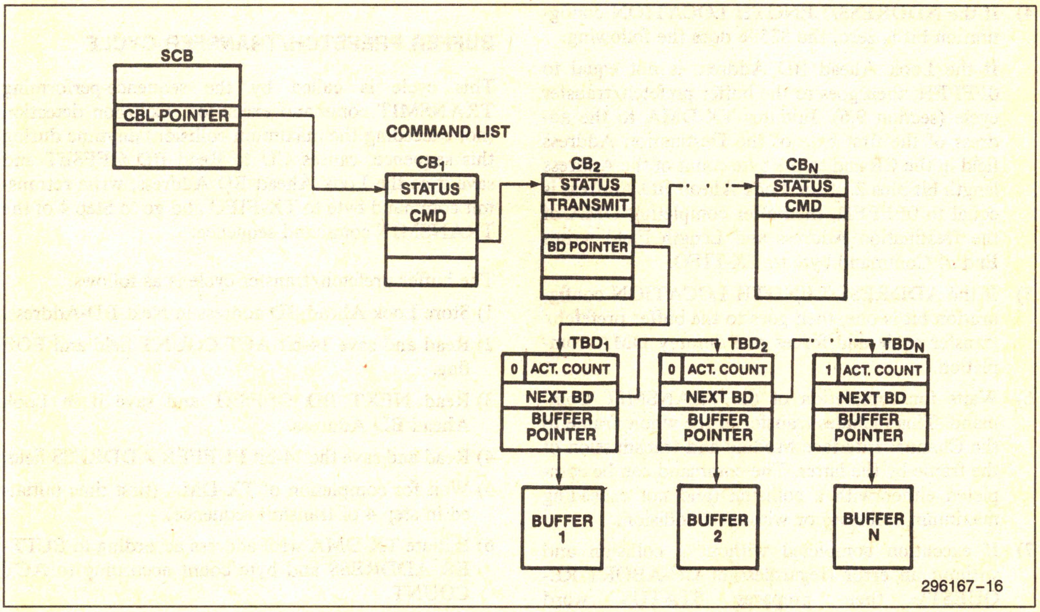


Figure 17. The Transmit Command Data Structure

While the CU is prefetching the address and byte count of one buffer, TX-DMA is transferring the previous buffer to the Transmit-Byte Machine via TX-FIFO. Completion of buffer transfer by TX-DMA, triggers the CU to initiate TX-DMA for the next buffer (if already prefetched) and to start the prefetch of the next buffer. The buffer prefetch/transfer cycle is terminated when a buffer, whose EOF bit is set, is transferred to TX-FIFO.

CU performs the following sequence during transmission:

1) Starts the Action Command.
2) Writes the TRANSMIT Command bit to TX-FIFO.
3) Reads BD OFFSET and saves it as the Look Ahead BD Address.
4) If the ADDRESS/LENGTH LOCATION configuration bit is zero, the 82586 does the following:

   If the Look Ahead BD Address is not equal to 0FFFFH, then goes to the buffer prefetch/transfer cycle (section 9.6). Initiates TX-DMA to the address of the first byte of the Destination Address field in the CB and to the byte count of the Address length bit plus 2. If the Look Ahead BD Address is equal to 0FFFFH, then after completing DMA of the Destination Address and Length Field writes End of Command byte to TX-FIFO.
5) If the ADDRESS/LENGTH LOCATION configuration bit is one, then goes to the buffer prefetch/transfer cycle and forces one dummy DMA completion.
6) Waits for completion of the TRANSMIT Command. This includes transfer of the whole frame to the Channel Interface Module and transmission of the frame by the latter. The command can be completed either with a collision (and not exceeding maximum collision) or without a collision.
7) If execution completed without a collision and without an error (regardless of CU-ABORT-REQUEST), then prepares STATUS word with C = 1, B = 0, OK = 1, A = 0, S6, S7 and NUM-COLL updated, and remaining bits zero.
8) If execution completed with CU-ABORT-REQUEST (and a collision or error) prepare STATUS word with C = 1, B = 0, OK = 0, A = 1 and NUM-COLL = 1.
9) If execution completed with a Collision (and not exceeding maximum collision), regardless of errors and without CU-ABORT-REQUEST, writes Retransmit command byte to TX-FIFO and returns to Step 3 of this sequence. This causes retransmission of the frame.
10) If the execution completed with an error (no collision or CU-ABORT-REQUEST), prepares STATUS word with C = 1, B = 0, OK = 0, A = 0 and the remaining bits updated.
11) Completes the Action Command.

Note that transmission is terminated by the CU performing one of the following steps; 7, 8, 9, or 10.

The Chart below summarizes which sequence step listed above the CU performs:

Collision	Error	Abort-Request	Step	Remarks
no	no	don't care	7	normal
no	yes	no	10	error
no	yes	yes	8	abort
yes	don't care	no	9	retransmit
yes	don't care	yes	8	abort

BUFFER PREFETCH/TRANSFER CYCLE

This cycle is called by the sequence-performing TRANSMIT command execution. Collision detection (not exceeding the maximum collision) anytime during this sequence, causes CU to Read BD OFFSET and save it as the Look Ahead BD Address, write retransmit command byte to TX-FIFO and go to Step 4 of the TRANSMIT command sequence.

The buffer prefetch/transfer cycle is as follows:

1) Store Look Ahead BD address in Next-BD-Address.
2) Read and save 14-bit ACT COUNT field and EOF flag.
3) Read NEXT BD OFFSET and save it in Look Ahead BD Address.
4) Read and save the 24-bit BUFFER ADDRESS field.
5) Wait for completion of TX-DMA (first time initiated in step 4 of transmit sequence).
6) Initiate TX-DMA with address according to BUFFER ADDRESS and byte count according to ACT COUNT.
7) If EOF bit is not set, repeat the cycle.
8) After DMA completion, write the End of Command byte to TX-FIFO.

FRAMING OPERATION

The Transmit-Byte-Machine maintains the following registers for construction of frames: Preamble pattern, SFD Field (pattern determined by End-of-Carrier/Bitstuffing configuration parameters), Source Address, CRC generator, Jam patterns (all ones).

After the Transmit-Byte-Machine reads the TRANSMIT command from TX-FIFO, a frame is constructed and transferred to the Transmit-Bit-Machine (see section 13) for bit transmission. The Transmit-Byte-Machine performs the following sequence:

1) Transfer Preamble bytes according to Preamble length configuration parameter minus one.
2) Transfer the SFD Field.
3) Start CRC calculation.
4) Read and transfer the Destination Address bytes from TX-FIFO (number determined by Address length).
5) If AL-Location configuration parameter is zero, transfer Individual Address as Source Address. Otherwise, read and transfer Source Address from TX-FIFO. If AL-Location = 1 and there are less than Address-Length bytes in TX-FIFO, a DMA underrun is forced.
6) Read and transfer all remaining bytes from TX-FIFO. These are the Length and Information fields.
7) Transfer CRC (calculated according to CRC 16/32 configuration parameter).
8) If the 82586 is configured to Bitstuffing, transfer one more flag.
9) If the 82586 is configured to Bitstuffing and to padding, transfer flag bytes to cause the number of transferred bytes (excluding Preamble and SFD Field) to exceed the configurable Slot Time divided by 8.

If a collision, underrun, or lost Carrier Sense occurred during transmission, the Transmit-Byte-Machine completes the transfer of the Preamble and transfers 4 bytes of the Jam pattern. If there is a collision, the retry counter will be incremented.

Jamming will not start before completing Preamble transmission.

If a collision is detected during transmission of the last 11 bits in the frame it will not result in jamming. If the collision is detected during transmission of the last bit or later, the collision will not be reported and retransmission does not take place. This may happen for an **invalid frame** which is shorter in length than the slot time.

Note that a DMA underrun cannot logically occur during the preamble because the serial subsystem generates its own preamble. Also, the 82586 is insensitive to carrier sense during the preamble.

After completing transmission, or collision, the Transmit-Byte-Machine passes bits 0–10 of the STATUS word to the CU.

8.7 TDR (Time Domain Reflectometer)

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify short or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR COMMAND FORMAT

TDR command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

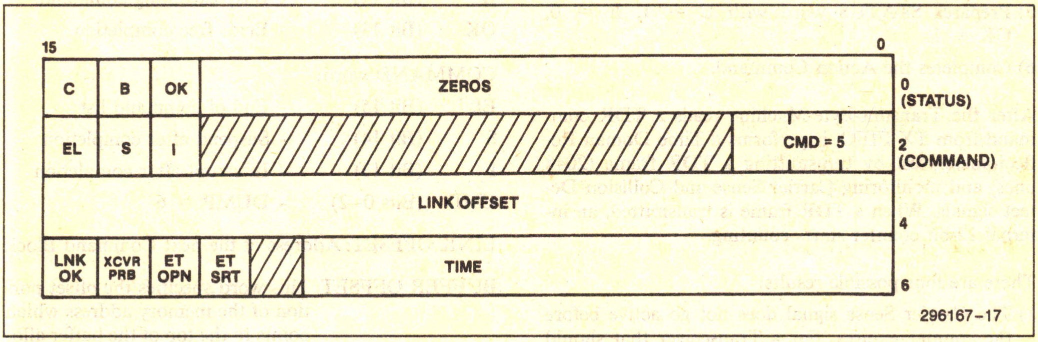


Figure 18. The TDR Command Block

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- TDR = 5

LINK OFFSET: Address of next Command Block

RESULT word:

LNK-OK	(Bit 15)	- No link problem identified
XCVR-PRB	(Bit 14)	- Transceiver Cable Problem identified (valid only in the case of a Transceiver that should return Carrier Sense during transmission).
ET-OPN	(Bit 13)	- Open on the link identified.
ET-SRT	(Bit 12)	- Short on the link identified (valid only in the case of a Transceiver that should return Carrier Sense during transmission).
TIME	(Bits 0–10)	- Specifying the distance to a problem on the link (if one exists) in transmit clock cycles.

DETAILED OPERATION OF TDR COMMAND

TDR command triggers the Time Domain Reflectometer test to be performed by the Transmit-Byte-Machine, see section 13.

The CU performs the following sequence:

1) Starts the Action Command.
2) Writes the TDR Command byte to TX-FIFO.
3) Waits for the Transmit-Byte-Machine to complete TDR test.
4) Writes results to RESULT word.
5) Prepares STATUS word with C = 1, B = 0, OK = 1.
6) Completes the Action Command.

After the Transmit-Byte-Machine reads a TDR command from TX-FIFO, it performs a Time Doman Reflectometer test by transmitting a TDR frame (2048 ones) and monitoring Carrier Sense and Collision Detect signals. When a TDR frame is transmitted, an internal 12-bit counter starts counting.

There are four possible results:

1) The Carrier Sense signal does not go active before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a problem on the cable between the 82856 and the Transceiver. For a Transceiver that should not return Carrier Sense during transmission, this is normal.
2) The Carrier Sense signal goes active and then inactive before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a short on the link.
3) The Collision Detect signal goes active before the counter expires. This means that the link is not properly terminated (an open).
4) The Carrier Sense signal goes active but does not go inactive and Collision Detect does not go active before the counter expires. This is the normal case and indicates that there is no problem on the link.

The distance to the cable failure can be calculated as follows:

Distance = TIME $\times$ (Vs/(2$\times$Fs))
where:

Vs - wave propagation speed on the link (M/s)

Fs - serial clock frequency (Hz)

Accuracy is plus/minus Vs/(2$\times$Fs)

Upon completion, the Transmit-Byte-Machine passes the RESULT word to the CU.

Note that the TDR frame does not contain the SFD field.

8.8 Dump

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

DUMP command includes the following fields:

STATUS word (written by the 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- DUMP = 6

LINK OFFSET: Address of the next Command Block

BUFFER OFFSET: This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

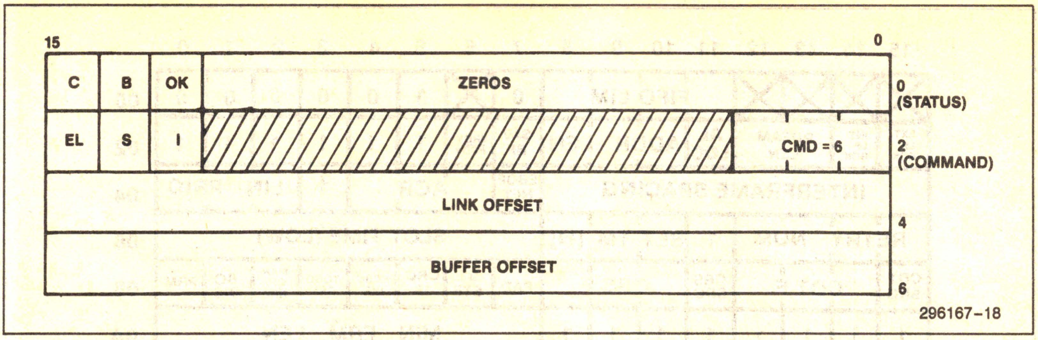


Figure 19. The DUMP Command Block

DUMP AREA FORMAT

Figure 20 shows the format of the DUMP area. The fields are as follows:

Bytes 00H to 0AH: These bytes correspond to the 82586 CONFIGURE command field.

Bytes 0CH to 11H: The Individual Address Register content. IAR0 is the Individual Address least significant byte.

Bytes 12H to 13H: Status word of last command block (only bits 0–13).

Bytes 14H to 17H: Content of the Transmit CRC generator. TXCRCR0 is the least significant byte. The contents are dependent on the activity before the DUMP command:

After RESET - 'All Ones.'

After successful transmission - 'All Zeros.'

After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.

After unsuccessful transmission, depends on where it stopped.

NOTE:
For 16-bit CRC only TXCRCR0 and TXCRCR1 are valid.

Bytes 18H to 1BH: Contents of Receive CRC Checker. TXCRCR0 is the least significant byte.

The contents are dependent on the activity performed before the DUMP command:

After RESET - 'All Ones.'

After good frame reception -

1) For CRC-CCITT - 0D1F0H.

2) For CRC-Autodin-II - C704DD7B.

After Bad Frame reception - corresponds to the received information.

After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

NOTE:
Any frame on the serial link modifies this register contents.

Bytes 1CH to 21H: Temporary Registers.

Bytes 22H to 23H: Receive Status Register. Bits 6, 7, 8, 10, 11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

Bytes 24H to 2BH: HASH TABLE.

Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of available bytes in the N + 1 buffer.

EL - The EL bit of the Receive Buffer Descriptor.

NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N + 1 Receive-Buffer Descriptor, i.e. the pointer to the N + 1 Receive Buffer.

CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame.

EL - The EL bit of the last buffer in the last received frame.

LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N + 2 Receive Buffer Descriptor.

NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N + 1 Receive Buffer Descriptor.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
X	X	X	X	FIFO LIM				0	X	0	0	0	0	0	0	00
EXT LP BCK	INT LP BCK	PREAM LEN		AL LOC	ADDR LEN			SAV BF	SRDY/ARDY	1	1	1	1	1	1	02
INTERFRAME SPACING								EBOF NET	ACR			1	LIN PRIO			04
RETRY NUM				1	SLT TM [H]			SLOT TIME (LOW)								06
CDT SRC	CDT F			CRS SRC	CRS F			PAD	BT STF	CRC 16	NCRC INS	TONO CRS	MAN CH/NRZ	BC DIS	PRM	08
1	1	1	1	1	1	1	1	MIN FRM LEN								0A
IAR 1								IAR 0								0C
IAR 3								IAR 2								0E
IAR 5								IAR 4								10
RCNT COL	0	TX OK	0	0	LST CRS	LST CTS	URN	TX DEF	SQET	MAX COL	0	COLL NUM				12
TXCRCR 1								TXCRCR 0								14
TXCRCR 3								TXCRCR 2								16
RXCRCR 1								RXCRCR 0								18
RXCRCR 3								RXCRCR 2								1A
TEMPR 1								TEMPR 0								1C
TEMPR 3								TEMPR 2								1E
TEMPR 5								TEMPR 4								20
1	0	RX OK	1	CRC ERR	ALN ERR	0	OVRN	SHRT FRM	NO EOP	1	1	1	1	1	1	22
HASHR 1								HASHR 0								24
HASHR 3								HASHR 2								26
HASHR 5								HASHR 4								28
HASHR 7								HASHR 6								2A
LNK OK	XCVR PRB	ET OPN	ET SRT	X	X	X	X	X	X	X	X	X	X	X	X	2C
1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	2E
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30

296167–19

Figure 20. The DUMP Area

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3A
																3C
0	0	0	0	0	0	0	0	0	0	0	0	0	ADR LEN			3E
EL		NXT RB SIZE														40
								NXT RB ADR (HIGH)								42
NXT RB ADR (LOW)																44
EL		CUR RB SIZE														46
LA RBD ADR																48
NXT RBD ADR																4A
CUR RBD ADR																4C
CUR RB EBC																4E
NXT FD ADR																50
CUR FD ADR																52
TEMPORARY																54
EOF		NXT TB CNT														56
BUF ADR																58
NXT TB ADR																5A
LA TBD ADR																5C
NXT TBD ADR																5E
EL	S	I											DUMP CMD CODE (110)			60
NXT CB ADR																62

296167-20

Figure 20. The DUMP Area (Continued)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
CUR CB ADR																64
																66
SCB ADR																68
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6C
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6E
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	70
																72
																74
				FIFO LIM				0		0	0	0	0	0	0	76
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7A
																7C
	0	0	0		SAV BF	END OF RDL	AL LOC	0	0	0	RU SUS RQ	0	0	0	0	7E
0		0	RU SUS FD				0		ABRT IN PRG	END OF CBL	CU SUS RQ	0	0	0	0	80
CX	FR	CNA	RNR	0		1	0	RU IDL	RU RDY	PU NRSRC	RU SUS	0	0	0	0	82
																84
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86
																88
																8A
																8C
																8E
0	0	0	0	0	0	0	0	BUF ADR PTR (HIGH)								90
BUF ADR PRT (LOW)																92
RCV DMA BC																94

296167-21

Figure 20. The DUMP Area (Continued)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
BR + BUF								ADR + H								96
0	0	0	0	0	0	0	0	RCV DMA ADR H								98
RCV DMA ADR L																9A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9C
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9E
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A2
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	SRDY/ARDY	A4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A8

296167-22

Figure 20. The DUMP Area (Continued)

CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but points to Nth Receive Buffer Descriptor.

CUR-RB-EBC: Current Receive Buffer Empty Byte Count. Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C = 1 and B = 0, then NXT-FD-ADR is the address of N+2 Receive Frame Descriptor (with B = C = 0) and is equal to the LINK-ADDRESS field in N+1 Receive Frame Descriptor.

CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N1 Receive Frame Descriptor (with B = 1, C = 0).

Bytes 54H to 55H: Temporary register.

NXT-TB-CNT: Next Transmit Buffer Count. Let N be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the Nth Transmit Buffer Descriptor.

EOF - Corresponds to the EOF bit of the Nth Transmit Buffer Descriptor. EOF = 1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the Nth buffer address.

LA-TBD-ADR: Look Ahead Transmit Buffer Descriptor Address. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the Nth Buffer Descriptor.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor N-1. Actually, it is the address of Transmit Buffer Descriptor N.

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request

END-OF-CBL (Bit 5) - End of Command Block List. If '1' indicates that DUMP-STATUS is the last command in the command chain.

ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.

RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume N is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of N + 1 Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.

RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.

RU-RDY (Bit 6) - Receive Unit in READY state.

RU-IDL (Bit 7) - Receive Unit in IDLE state.

RNR (Bit 12) - RNR Interrupt In Service bit.

CNA (Bit 13) - CNA Interrupt In Service bit.

FR (Bit 14) - FR Interrupt In Service bit.

CX (Bit 15) - CX Interrupt In Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AL-LOCation configuration bit.

1) If AL-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.

2) If AL-LOCation = 1 then it contains the size of the next Receive Buffer.

BR + BUF-PTR + 96H - Sum of Base Address plus BUF-PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AL-LOCation configuration bit.

1) If AL-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.

2) If AL-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.

The following notes apply to all the information of the dump area:

1) The listed pointers are meaningful only if the 82586 has completed the prefetch.
2) All Transmit Pointers refer to the last executed TRANSMIT Command.
3) The Receive Pointers are meaningful only if they exist, namely a frame was received into the existing memory structure.
4) The following nomenclature has been used in the DUMP table:
   0 - The 82586 writes zero in this location.
   1 - The 82586 writes one in this location.
   X - The 82586 writes zero or one in this location.
   ///- The 82586 copies this location from the corresponding position in the memory structure.

DETAILED OPERATION OF DUMP COMMAND

Configuration parameters and contents of other registers are transferred from the Channel Interfaced Module via RCV-FIFO by Receive Unit to memory.

The Command Unit performs the following sequence:

1) Starts Action Command.
2) Writes DUMP command byte to TX-FIFO.
3) Waits for completion of DUMP.
4) Prepares STATUS word with C= 1, B = 0, OK = 1.
5) Completes Action Command.

The Receive Unit performs the following sequence.

1) Writes the word with the FIFO-Threshold configuration field to memory.
2) For all data written by the Channel Interface Module to RCV-FIFO: Reads 2 bytes from FIFO, assembles them into a word and writes it to the next location in memory buffer.
3) Writes 64 CU and RU registers to the remaining space in memory buffer.
4) Notifies CU that DUMP completed.

The Channel Interface Module passes 46 bytes of internal registers to RU and notifies CU on completion.

NOTE:
This is the only Action Command involving the RU.

8.9 Diagnose

The DIAGNOSE Command triggers an internal self test procedure of backoff related register and counters.

DIAGNOSE COMMAND FORMAT

The DIAGNOSE command includes the following:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed (see section 8.1)
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
FAIL	(Bit 11)	- Indicates that the self test procedure failed

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0–2)	- DIAGNOSE = 7

LINK OFFSET: Address of next Command Block

OPERATION DETAILS OF DIAGNOSE COMMAND

CU triggers the self test procedure of the Channel Interface Module. It performs the following sequence:

1) Starts Action Command.
2) Writes DIAGNOSE command byte to TX-FIFO.
3) Waits for command completion.
4) If diagnose succeeded, then prepares STATUS word with C = 1, B = 0, OK = 1, FAIL = 0; otherwise, prepares STATUS word with C = 1, B = 0, OK = 0, FAIL = 1.
5) Completes Action Command.

The Channel Interface Module performs the self test procedure in two phases: Phase 1 tests the counters and Phase 2 tests the trigger logic.

During Phase 1, Free Run, Exponential Backoff Timeout, Slot Time and Collision Counters are checked.

The test is performed in the following steps:

1) All counters are RESET simultaneously.
2) Start counting.
3) Stop counting when the Free Run Counter (10 bits), and Exponential Backoff Counter (10 bits), wrap from 'All Ones' to 'All Zeros.' Simultaneously, the Slot Time counter switches from 01111111111 to 10000000000, and the collision counter (4 bits) wraps from 'All Ones' to 'All Zeros.'
4) Phase 1 is successful if the 10 least significant bits (when applicable), of all four counters are 'all zeros.'

During Phase 2, the test is performed in the following steps:

1) Reset Exponential Backoff Shift Register, and all counters.
2) Temporarily configure internally, Exponential Backoff logic according to the following:

   SLOT-TIME = 0BH

   LIN-PRIO = 02H

   EXP-PRIO = 01H

   BOF-MET = 00H
3) Emulate internally, transmission and collision.
4) If the most significant bit of Exponential Backoff Shift Register is 0, then go to step 3.
5) Check Mask Logic output for being 'All Ones' (Free Run Counter is 'All Ones' at this point and Exponential Backoff Shift Register is also 'All Ones').

If Step 5 is successful, then a 'Passed' status is returned otherwise, 'Failed' status is returned.

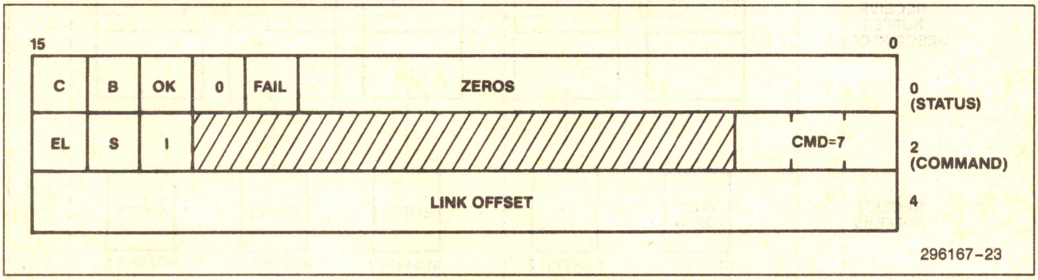


Figure 21. The DIAGNOSE Command Block

9.0 FRAME RECEPTION

Section 7 described how the user can control frame reception. This section presents the details of receiving and storing frames in memory.

9.1 Receive Frame Area (RFA)

The Receive Frame Area, RFA, is prepared by the host CPU. The 82586 places data into the RFA as frames are received. The RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. The RFA-OFFSET field of the SCB points to the first FD of the chain; the last FD is identified by the End of List flag (EL). See Figure 22.

In general, the incoming frame length is unknown beforehand. Rather than allocating buffers with a length greater than the largest expected frame, the 82586 makes it possible to store frames in a sequence of small buffers, which are chained into complete frames. Buffers are chained via Receive Buffer Descriptors (RBD), which point to a single buffer.

9.2 Frame Descriptor (FD) Format

The FD (see Figure 23) includes the following fields:

STATUS word (**written** by the 82586):

C	(Bit 15)	- Completed storing a frame.
B	(Bit 14)	- The 82586 knows about this FD and is ready to use it.
OK	(Bit 13)	- Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.
S11	(Bit 11)	- CRC Error: The received frame experienced a CRC error and was *well aligned.*

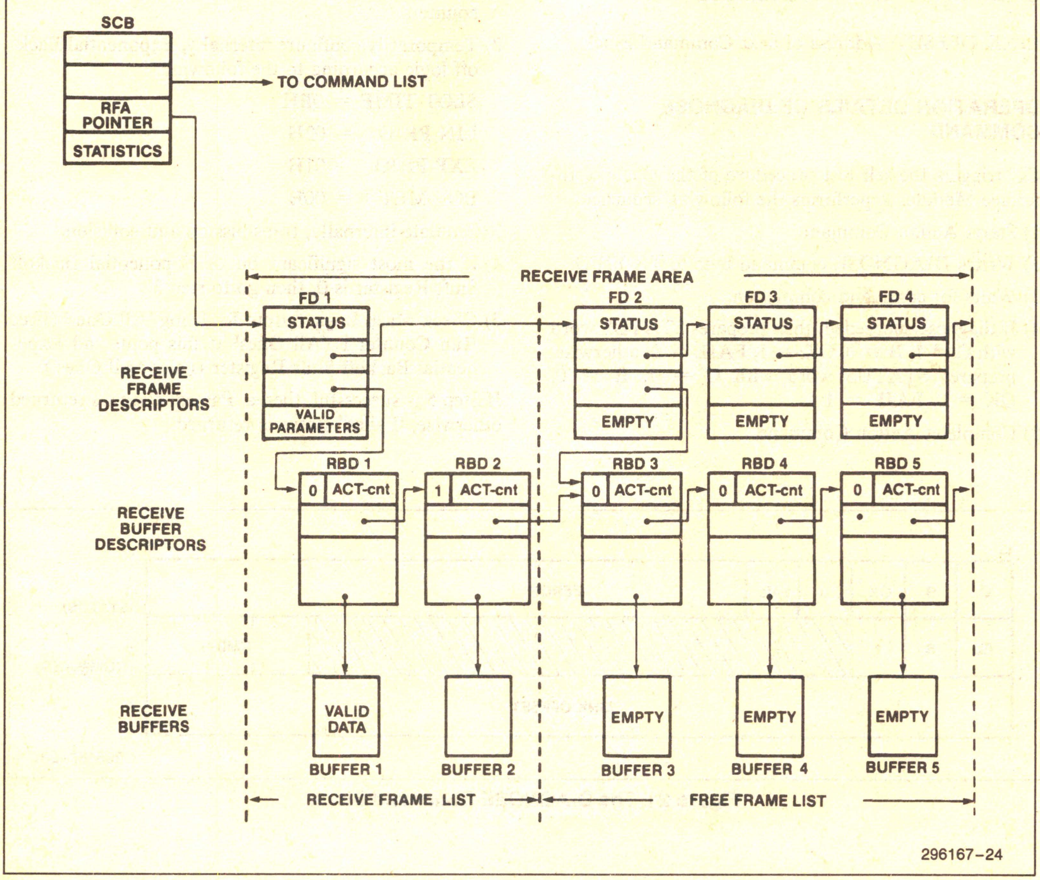


Figure 22. The Receive Frame Area

S10 (Bit 10) - Alignment Error: The received frame experienced a CRC error and was *misaligned.* The 82586 does not report misaligned frames with a correct CRC.

S9 (Bit 9) - The RU ran out of resources during reception of this frame.

S8 (Bit 8) - RCV-DMA overrun.

S7 (Bit 7) - The received frame had fewer bytes than configured Minimum Frame Length.

S6 (Bit 6) - No EOF flag detected (only when configured to Bitstuffing).

COMMAND word:

EL (Bit 15) - The last FD on the list.

S (Bit 14) - The RU should be suspended after receiving this frame.

LINK OFFSET: Address of the next FD on the list.

RBD-OFFSET (initially prepared by the CPU and later may be updated by the 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means that there are no RBDs associated with this FD.

DESTINATION ADDRESS (**written** by the 82586): Contains Destination Address of received frame. The length in bytes is determined by the Address Length configuration parameter.

SOURCE ADDRESS (**written** by the 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS.

LENGTH FIELD (**written** by the 82586): Contains the 2 byte Length Field of the received frame.

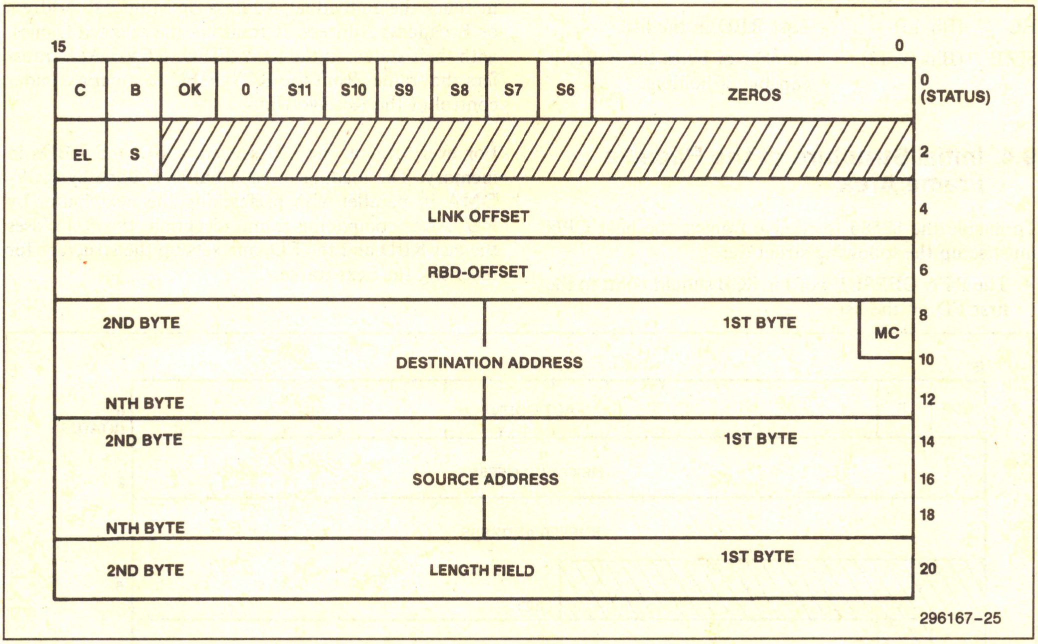


Figure 23. The Frame Descriptor (FD) Format

9.3 Receive Buffer Descriptor Format

The Receive Buffer Descriptor (RBD) holds information about a buffer; size and location, and the means for chaining RBDs (forward pointer and end-of-frame indication).

The Receive Buffer Descriptor contains the following fields:

STATUS word (**written** by the 82586):

EOF (Bit 15) - Last buffer holding the received frame.

F (Bit 14) - ACT COUNT field is valid.

ACT COUNT (Bits 0–13) - Number of bytes in the buffer that are actually occupied.

NEXT RBD OFFSET (**written** by the CPU): Address of the next BD in the list of BDs.

BUFFER ADDRESS (**written** by the CPU): 24-bit absolute address of buffer.

EL/SIZE (**written** by the CPU):

EL (Bit 15) - Last RBD in the list.

SIZE (Bits 0–13) - number of bytes the buffer is capable of holding.

9.4 Initial Structure of the Receive Frame Area

To enable the 82586 to receive frames, the host CPU must setup the following structure:

- The RFA OFFSET word in SCB should point to the first FD on the list.
- The LINK OFFSET of each FD in the list should point to the next FD.
- The EL bit in the last FD should be set.
- The RBD OFFSET of the first FD in the list should point to the first RBD in the RBD list. The RBD OFFSET in the remaining FDs must be 0FFFFH; it will be later loaded by the 82586 with the address offset of the first RBD assigned to the frame (unless the No Resources state is reached).
- The NEXT RBD OFFSET of each RBD in the list should point to the next one.
- The EL bit in the last RBD should be set.
- The BUFFER ADDRESS and SIZE fields in each BD should indicate the location and available space in a buffer.

9.5 Detailed Operation of Receiving a Frame

The Channel Interface Module selects the frames destined for the 82586 according to the Destination Address of the frames passing on the link. A frame is selected if it is at least 6 bytes long and its address matches the Individual Address or Multicast Address or Broadcast Address. It transfers the selected frames, with their status, to the RCV-FIFO. RCV-DMA transfers the frames from the RCV-FIFO to memory under control of the Receive Unit.

For every frame, the RU sets up a FD and RBDs in memory. The loading of each buffer is done by RCV-DMA in parallel with prefetching the next buffer by RU. After completing frame reception, the RU closes the last RBD and the FD, and sets up the structure for receiving the next frame.

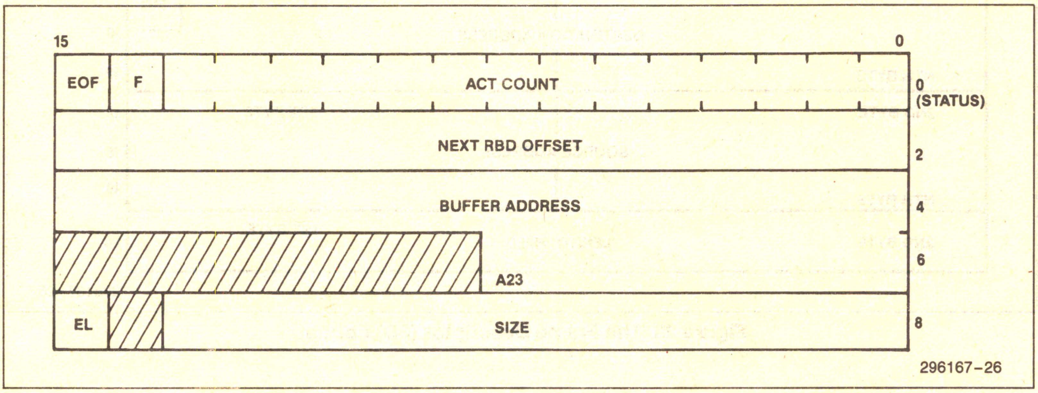


Figure 24. The Receive Buffer Descriptor (RBD) Format

SETUP NEXT FD

The RU performs the following sequence to setup a FD:

1) Writes a word with B = 1, and the remaining bits set to zero, to STATUS word of the next FD.
2) If AL-Location configuration parameter is zero, initiates RCV-DMA with the address of the first byte of the DESTINATION ADDRESS and a byte count of twice the Address Length configuration plus two.
3) Saves the value of the NXT FD ADR in the CUR FD ADR. This operation is done internally and is transparent to the BUS.
4) Reads COMMAND word of the current FD and saves EL and S bits internally.
5) Reads LINK OFFSET word of FD and saves it in the address of NEXT FD.
6) If a buffer has been prefetched, writes the offset part of its address to the RBD-OFFSET field of the current FD.
7) If a buffer is not available, reads the RBD OFFSET word from the current FD and saves it in the Look Ahead RBD address. If it is all ones, sets internal END-OF-LAST-BUFFER flag.
8) If AL-Location is one, forces a RCV-DMA completion.
9) Goes to the buffer prefetch/transfer cycle.

BUFFER PREFETCH/TRANSFER CYCLE

The RU prefetches a buffer according to the Look Ahead BD address. It also maintains two internal flags for beginning and end of a cycle.

1) If a buffer is already full, writes F = 1 and current size to STATUS word of current BD.
2) Saves address of next RBD in current RBD and address of the Look Ahead RBD in next RBD. Saves Next Size in current size.
3) If END-OF-LAST-BUFFER is set, go to 7.
4) Reads RBD-OFFSET from Next BD and saves it in Look Ahead BD address.
5) Reads 24-bit BUFFER ADDRESS from Next BD and saves it internally.
6) Reads EL SIZE word from Next BD and saves it in internal EL bit and Next size.
7) Waits for completion of RCV-DMA.
8) IF END OF LAST BUFFER is set, starts discarding incoming frame and quits.
9) Initiates RCV-DMA with Next Buffer Address and Next Size as byte count.
10) If the EL bit is set, sets END-OF-LAST-BUFFER.
11) Repeats the cycle.

CLOSE FRAME

When the RU reads an End of Frame from RCV-FIFO (indicating the end of a received or discarded frame), it performs the following sequence to close the frame:

1) Reads the status (2 bytes) from RCV-FIFO and saves it internally.
2) If the RU is not in the READY state, it goes straight to Completion of Reception sequence.
3) If the frame status indicates that there is an error (including short frame) and the Save Bad Frame configuration parameter is zero, or ADDRESS and LENGTH fields in FD are not full, it reuses the FD; Backs up Current FD to Next FD, discards pointer to the current RBD, and goes straight to step 9.
4) If the frame included only ADDRESS and LENGTH fields (i.e. no buffers used) writes 'all ones' to BD-OFFSET of Current FD.
5) If the frame filled at least part of a buffer, writes to STATUS word of the current RBD: EOF = 1, F = 1, and actual byte count.
6) If the RU ran out of resources during reception of this frame, writes to STATUS word of Current FD: C = 1, B = 0, OK = 0, S9 = 1, and the remaining bits from the Receive-Byte-Machine.
7) If the RU did not run out of resources, write to STATUS word of current FD: C = 1, B = 0, and the remaining bits from the Receive-Byte-Machine.
8) Requests FR interrupt.
9) If there was no RU-START-REQUEST or RU-SUSPEND-REQUEST or S bit and the RU did not run out of resources, it sets up a new FD (see section 10.2).
10) Goes to complete reception of frame.

COMPLETION OF RECEPTION

Reception completion occurs when the RU encounters an end of frame (after closing it) regardless of its state (i.e. also when discarding a frame). The decision on how to proceed is determined by the following: EL bit, S bit, RU-START-REQUEST, RU-SUSPEND-REQUEST and whether the RU ran out of buffer descriptors.

The following sequence is performed by the RU at Reception completion:

1) If the RU ran out of buffers or Frame Descriptors during current frame reception, then: change state to NO-RESOURCES, request an RNR interrupt, clear RU-SUSPEND-REQUEST and (internal) S bit, and start discarding.
2) If the received frame was longer than the Minimum Frame Length, update all four statistics registers (see details in section 8.6). Note that the RSCERRS statistics register is updated if the RU is in the NO-RESOURCES state or if it ran out of buffers during this frame.
3) If the S bit of current frame or RU-SUSPEND-REQUEST is set and RU-START-REQUEST is not, then: if the RU is in the READY state, request an RNR interrupt, and change state to SUSPENDED.
4) If RU-START-REQUEST is set and the S bit not, then: change the RU state to READY and perform Setup new FD.
5) If the RU-START-REQUEST is set and also S bit is set, perform Setup new FD and change state to SUSPENDED.
6) If the RU state is different from READY, start discarding.
7) If the RU is in the READY state or just exited it, then perform the following:
   a) Set respective INTERRUPT-IS flags.
   b) Deactivate hardware Interrupt signal.
   c) Clear interrupt request flags.
   d) Update SCB STATUS word according to the new state and INTERRUPT-IS flags.
   e) Activate hardware Interrupt signal.

If a command is being executed by the serial machine (e.g. Configure, IA Set-up, Dump, etc.) it will ignore incoming frames during command execution. Thus, frames can be lost during this time. Conversely, if a frame is being received, the serial machine will wait until the entire frame has been received before the command is executed. Section 10.5 describes simultaneous Transmit and Receive operation.

OPERATION OF RECEIVE-BYTE MACHINE DURING RECEPTION

For every frame that arrives from the Receive-Bit-Machine (after it strips the Preamble and detects the SFD field) the Receive-Byte-Machine decides, according to the Destination Address, whether to accept the frame (see section 12). If the frame is accepted, the Receive-Byte-Machine transfers the Destination Address, Source Address, Length Field and Data field to RCV-FIFO. If it is not accepted, nothing enters RCV-FIFO. If RCV-FIFO overruns, the Receive-Byte-Machine keeps overwriting the input buffer of RCV-FIFO (without affecting the bytes that are already in RCV-FIFO). As soon as space is available, new bytes enter RCV-FIFO. The Receive-Bit-Machine strips the CRC and Padding (if applicable) and prepares the STATUS word with bits indicating: occurrence of any error, CRC or alignment error, DMA overrun, frame too short, or absence of EL flag (in Bitstuffing). The Receive-Byte-Machine enters the STATUS word in two consecutive bytes to the RCV-FIFO.

10.0 BUS INTERFACE

The 82586 can operate on the host CPU local bus as a bus master with shared status lines. Since status lines are in common, all local bus resources (Clock Generator, 8288 Bus Controller, Address Latches and Data Transceivers) can be shared by the CPU and the 82586. To facilitate minimum component count designs, the 82586 bus interface has been optimized for use with an 80186 16-bit microprocessor. The 82586 also interfaces to the 8086/88 and other processors. Dual port memory configurations can be used to minimize the impact of 82586 bus bandwidth requirements on system performance.

When operating in Maximum Mode (MN/$\overline{\text{MX}}$ pin strapped to logic zero) pins $\overline{\text{S0}}$ and $\overline{\text{S1}}$ are connected to the 8288 bus controller, which in turn generates all control signals for the system interface. In this case, the full address range of 16 megabytes is available.

In Minimum Mode (MN/$\overline{\text{MX}}$ pin strapped to logic one), the 82586 generates ALE, $\overline{\text{DEN}}$, DT/$\overline{\text{R}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals. An address space of 4 megabytes is available.

The system bus is acquired and released using HOLD/HLDA protocol directly interfaced to the 80186. External hardware can be used to convert HOLD/HLDA to $\overline{\text{RQ}}/\overline{\text{GT}}$ protocol suitable for interfacing to the 8086 CPU.

When interfacing to a 16-bit bus, data is multiplexed with the 16 lower address pins. Examples of this mode are systems with 8086, 80186, 80286 or other 16-bit microprocessors. On an 8-bit data bus, data is multiplexed with the 8 lower address pins. Examples of this mode are systems with 8080/8085, 8088 or other 8-bit microprocessors.

All bus timing and loading specifications are consistent with those of the 80186 system. Bus timing and loading specifications are given in the 82586 Data Sheet.

The 82586 CLK signal input must receive MOS level inputs. The 82586 can operate with an 80186 or a 8086-1 using an 8 MHz 50% Duty Cycle clock (full performance), or with a 5 MHz 8086 using a 5 MHz 30% Duty Cycle clock (reduced performance).

10.1 Memory Addressing and Organization

The 82586 addresses memory as a linear sequence of 16 megabytes. It establishes transfers as 16-bit words when it is used with an 8086/80186/80286 or as 8-bit bytes using an 8088 or other 8-bit processors.

As a Master Peripheral with the 8088, the 82586 treats memory as a single bank (D7–D0) of 1M 8-bit bytes addressed by address lines A19–A0. Address lines A23–A20 are not used, $\overline{BHE}$ is strapped HIGH.

When operating with the 8086/80186, the 82586 treats memory as a high bank (D15–D8) and a low bank (D7–D0) of 512K 8-bit words addressed in parallel by A19–A1. The 82586 performs Reading/Writing from/to full word locations only. The $\overline{BHE}$ line is always LOW. Word transfers are restricted to even addressing, although the 82586 does not enforce pin A0 to LOW. When attempting to write to an odd location (i.e. when an odd buffer address is set), the odd address will be output to the address lines, but a full word will be output to the data lines, in which the low byte will be undefined. The manner in which line A0 being HIGH is treated is system dependent. The data words flow through pins D15–D0.

The Chart below summarizes the operation of $\overline{BHE}$ and A0 signals:

$\overline{BHE}$	A0
0	0 whole word
0	1 system dependent
1	0 lower byte to/from even address
1	1 upper byte to/from odd address

When operating with the 80286, the full 16 megabyte address space is available. It has the same characteristics and restrictions as the 8086/80186.

The 82586 accesses all control information from a segmented memory structure. The 16-bit pointers residing in the various structures are used as an offset to a 24-bit base location. Hence, all FD, RBD, CB, TBD, SCB structures must reside in 64K byte boundaries. Data buffers may reside anywhere in address space, and are directly addressed as a 24-bit address.

10.2 Bus Operation

Each memory transfer cycle consists of at least four CLK cycles. These are referred to as t1, t2, t3 and t4. The address is generated by the 82586 beginning at t1, and data transfer occurs on the bus during t2 through t3. In the event that a READY indication is not received from the addressed memory, WAIT states (tW) are inserted between t3 and t4. Each inserted WAIT state has the same duration as a CLK cycle.

The 82586 outputs status ($\overline{S1}$ and $\overline{S0}$) to provide type-of-memory-cycle information to the 8288 Bus Controller in Maximum Mode or in Multibus Configuration. The 8288 generates memory read and write commands, and issues control signals to the address buffers and data transceivers. The 82586 provides $\overline{RD}$, $\overline{WR}$ and ALE signals for memory transfers in Minimum Mode. A multi-master system bus can be constructed using the 8289 Bus Arbiter. The key arbiter inputs are the same as for the 8288: local status lines $\overline{S1}$ and $\overline{S0}$.

Operation of the 82586 is structured so that all command, status and data flow is via memory. Therefore, the S2 status line of the 8086 and 80186 families is not required and is not provided by the 82586. With the exception of SCP, which is always read from location 0FFFFF6H, there are no transfers to/from a fixed address system resources (I/O memory).

READ

The read cycle begins at t1 by generating the address. The memory read command signal (MRDC from the 8288) is asserted at t2. This command causes the addressed device to enable its data bus drivers onto the system bus. Some time later, valid data will drive the READY line HIGH. The data will be sampled by the 82586 at t4. If the READY line is still LOW in t3, then tW cycles are inserted between t3 and t4 until the READY line goes HIGH. When the MRDC signal returns to the HIGH level, the addressed device will again tri-state its data bus drivers. If a transceiver (8286/8287) is required to buffer the local bus, the direction (DT/$\overline{R}$) and enable ($\overline{DEN}$) controls are provided by the 8288 Bus Controller (or from the 82586).

WRITE

A write cycle begins by generating the address during t1. At t2 the processor generates data to be written. This data remains valid until the middle of t4. During t2, t3 and tW, the advanced memory write command (AMWTC) from the 8288 is asserted, while the normal memory write command (MWTC) is asserted during t3 and tW only. MWTC is used by older memories requiring valid data prior to the write command.

SYNCHRONIZATION WITH LOWER SPEED MEMORIES

The 82586 uses the standard READY mechanism to work with memories having long access times.

As bus state t3 is reached, the READY signal is tested. While the READY signal is LOW, wait states (tW), are added and all control signals are stretched accordingly. While in wait states, (tW), the READY signal is tested at every clock cycle. The first time that READY is found to be HIGH, state t4 is entered, finishing the bus cycle.

In a synchronous environment, the user is able to guarantee setup and hold times for the READY signal related to the system clock. Synchronous READY input is chosen and the number of wait states is predictable.

For asynchronous environments, the user is not required to provide precise setup and hold times for the READY signal. A high speed resolution circuit synchronizes the asynchronous READY input internally to the 82586. Setup and hold times for the asynchronous READY input are not required for correct operation but for recognition at a certain clock. The number of wait states is therefore not always predictable.

Synchronous READY input is sampled by the 82586 at the beginning of t3. Setup and hold time requirements refer to the falling edge of the CLK when entering t3 state.

Asynchronous READY input is synchronized by the 82586 in the middle of t2. Setup and hold times refer to the rising edge of the CLK while in state t2.

In Minimum Mode, the 82586 has one input READY pin, SRDY/$\overline{\text{ARDY}}$. This pin is software programmable to work as either synchronous or asynchronous. Upon RESET, the 82586 automatically defaults to asynchronous mode. The CONFIGURE command is used to program the circuitry to synchronous or asynchronous READY.

In Maximum Mode the SRDY/$\overline{\text{ARDY}}$ pin behaves exactly as in Minimum Mode. In addition, the READY pin provides an alternate synchronous READY connection. Internally, the 82586 performs a logic OR function, between the SRDY/$\overline{\text{ARDY}}$ pin and the READY pin.

10.3 Bus Acquisition

The system bus is acquired and released by means of the HOLD/HLDA protocol.

When the 82586 needs the bus, it activates the Hold signal. Upon receiving the HLDA, it initiates bus transfers. At the end of bus transfers, it relinquishes the bus by deactivating the HOLD and, subsequently, the host takes away the HLDA.

Figure 25A shows the number of clocks the 82586 requires after getting an HLDA before it starts a bus transfer.

The 82586 does not hold the bus unless it is doing memory transfers, with one exception. At the end of a received frame, the 82586 starts post-frame processing, which includes posting Status for the recently received frame and getting ready for the next frame to be received. All these processes have to be completed such that Interframe Spacing requirements (9.6 μs for IEEE 802.3) are met. To be able to do that, the 82586 may hold the bus for 173 clock periods (Typical value for AL-LOC = 0, SAV-BF = 0). The chip, however, does not really use the bus for entire 173 clock periods for post-frame processing. About 66 of the 173 clock periods are not used (no Read or Write operations). The chip simply holds the bus for 173 clock periods in anticipation of the next frame which may come in after the IFS time (9.6 μs) in the worst case (see Figure 25). If the RU finds that the next frame is not coming in immediately, then the 82586 will release the bus after 173 clock periods. If the next frame is coming in, the chip will hold the bus even longer and transfer the first part of the next frame into memory. In this case, receive data DMA will be interleaved with receive control DMA. The system design must not interpret post-frame processing as a fault condition.

The CPU can force the 82586 off the bus by removing HLDA. The 82586 will complete the current bus transfer and will relinquish the bus within a maximum of four clock cycles in Word mode or eight clock cycles in Byte mode, see Figures 26A and 26B. However, if the 82586 still has some pending transactions, the Hold will be activated again after 1 clock cycle (minimum hold drop time is 1 clock cycle).

READY TIMEOUT

The 82586 Ready input is sampled during DMA operations. If a system error (e.g. RAM error) causes the Ready to not be returned to the 82586, the 82586 will indefinitely hold on to the bus. A removal of HLDA will not make the 82586 release the bus under these conditions (typically, the 82586 gives up the bus within 3 clock cycles of HLDA going inactive). The system designer must incorporate some external logic to detect the 'Ready Absence' condition and reset the 82586 in such an event. In addition, the CPU must initiate some recovery procedures (e.g. RAM test) to isolate the problem and follow up with remedial procedures.

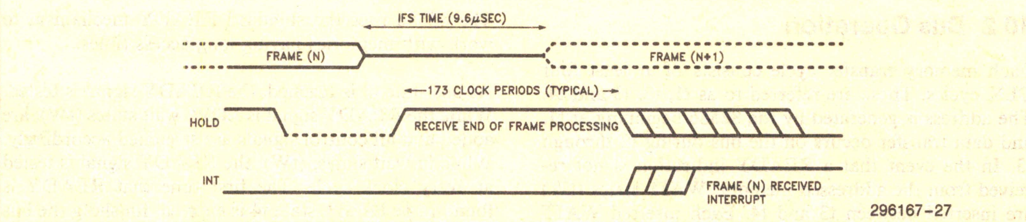


Figure 25. Receive End of Frame Processing

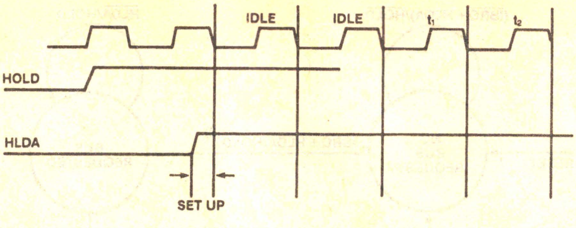


Figure 25A. HOLD/HLDA: Bus Transfer Timing

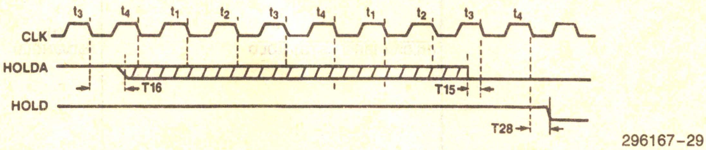


Figure 26A. 82586 Byte Mode Bus Relinquish Timing

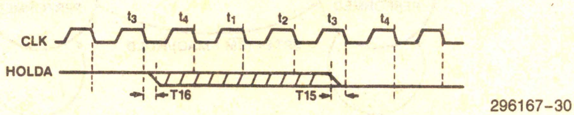


Figure 26B. 82586 Word Mode Bus Relinquish Timing

DETAILS OF THE HOLD/HLDA PROTOCOL

The state diagram of Figure 27 defines the states and excitations of the bus acquisition protocol.

STATES:

NO BUS REQUEST: The 82586 is not requesting the bus via HOLD, because Internal Bus Request (IBRQ) is not pending, or when the IBRQ arrived, HLDA level was HIGH.

BUS REQUESTED: The 82586 has issued HOLD (bus request). HLDA is LOW, i.e. bus was not yet granted.

BUS MASTER/TRANSFERS PERFORMED: The 82586 owns the bus and performs transfers.

BUS MASTER/NO TRANSFERS PERFORMED: The 82586 owns the bus, but does not use it for transfers. This is used by the 82586 to lock the bus for a few system clock cycles for READ/MODIFY/WRITE operations, or when the 82586 knows that it will need the bus soon (e.g. when buffer switching occurs).

EXCITATIONS:

RESET: Software or Hardware Reset initializes the machine.

IBRQ: Internal Bus Request. Any internal demand to perform bus transfers on the bus: Receive or Transmit DMA, CU or RU.

STD: Special state different from t1, t2, t3, tW and t4, for which the 82586 masters the bus, but no transfers are performed (ALE, $\overline{RD}$, $\overline{WR}$ are inactive in Minimum Mode or $\overline{S0}$, $\overline{S1}$ are inactive in Maximum Mode).

SPT4: State prior to t4. May be t3 for zero wait states or the last tW for any number of wait states.

BM: Configured to 8-bit bus width.

MA0: Memorized A0. A0 was true during the last t1 (Odd-Address bus cycle).

The following algorithm is followed regarding bus acquisition.

1) RESET sets the 82586 to the NO BUS REQUEST state.
2) The internal bus request raises HOLD output and switches the state to BUS REQUESTED. HLDA input, being LOW, enables this switch of state. The reason for this condition is to differentiate HLDA being HIGH as a result of the previous bus cycle from newly generated HOLD acknowledge.
3) Appearance of HLDA sets the state to BUS MASTER/TRANSFERS PERFORMED. The bus is acquired and read/write operations are started.

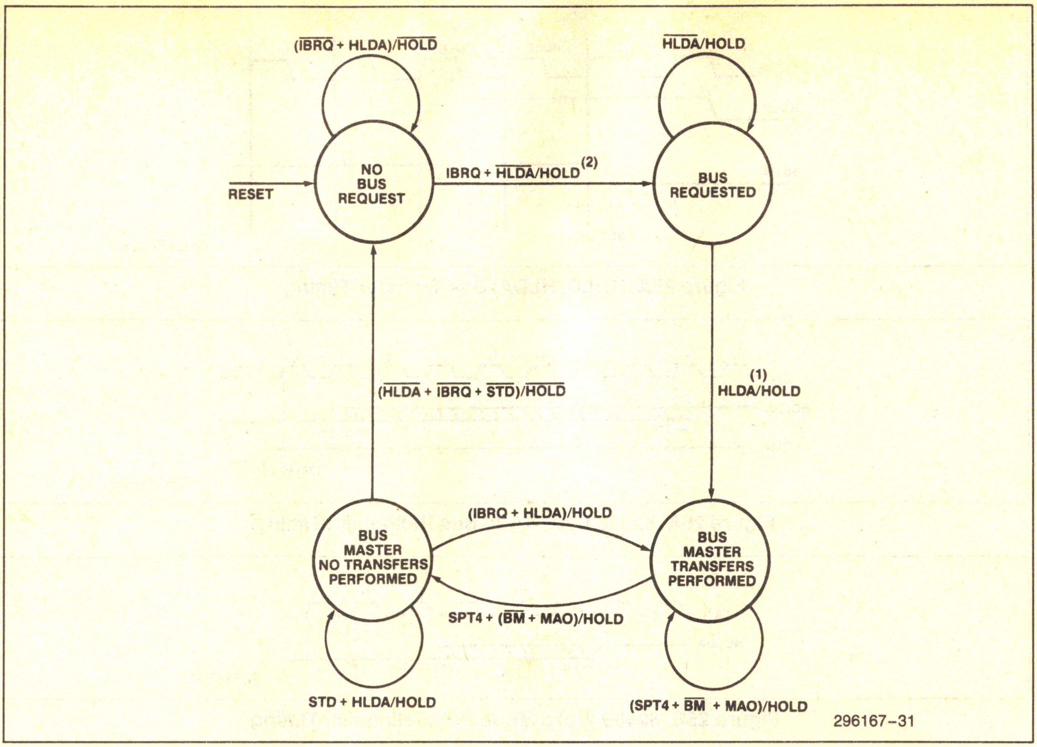


Figure 27. The HOLD/HLDA Handshake

4) Prior to entering bus state t4, the 82586 evaluates the next action: If HLDA input is active and there are pending internal bus requests, the state remains BUS MASTER/TRANSFERS PERFORMED. Back to back bus cycles result. If HLDA is deactivated or no more internal bus requests are present, the 82586 switches to NO BUS REQUEST state.

5) State BUS MASTER/NO TRANSFERS PERFORMED is entered for all cases that no internal bus request exists but performance considerations require holding the bus until the request shows up.

10.4 FIFO-Threshold Mechanism

The on-chip Receive and Transmit FIFOs provide buffering between the serial channel and system bus. On the system side, data is written into the Transmit FIFO or read from the Receive FIFO, using the 82586 DMA data channels. The Transmit DMA bursts fill the Transmit FIFO at the bus data rate. The 82586 Channel Interface Module pulls bytes of data from the Transmit FIFO and transmits to the link at the serial bit rate. The Receive FIFO operates analogously, filled by the Channel Interface Module and emptied by bursts for Receive DMA channel.

The transactions involved in bus acquisition and release imply overhead resulting in losing system clocks due to signal propagation delays and sample time requirements. For many short DMA bursts, up to 5 clocks are lost due to this switching. On the other hand, long DMA bursts imply that bus masters, other than 82586, may experience long delays in bus acquisition. Thus, an optimal FIFO-Threshold setting exists.

The 82586 provides a means for choosing the optimal setting required by the user's application, using the FIFO-Threshold configuration parameter.

TRANSMIT FIFO OPERATION

Starting from an empty Transmit FIFO, the transmit DMA starts reading memory locations, and pushing bytes into Transmit FIFO. The Channel Interface Module starts transmitting on the link, pulling bytes from the Transmit FIFO, at the serial bit rate. Because the data system (byte) rate is faster than the serial's, the Transmit FIFO will eventually become full. At this point, the DMA burst stops.

Meanwhile, the Channel Interface Module continues to transmit, thus emptying the FIFO. When the number of bytes in the Transmit FIFO equals the configured FIFO-Threshold, the system bus is requested for a new DMA burst. If the bus is not acquired fast enough, the Transmit FIFO will be emptied by the Channel Interface Module and a Transmit FIFO underrun occurs. The optimal value for the FIFO-Threshold is such that it guarantees bus acquisition before underrunning the FIFO, and at the same time provides the longest possible burst.

When the bus is acquired, the new Transmit DMA burst continues until the FIFO is full. This mechanism can be viewed as an hysteresis process, with the upper bound Transmit FIFO full and the lower bound the programmed FIFO-Threshold.

RECEIVE FIFO OPERATION

At the beginning of frame reception, the Channel Interface Module writes into the Receive FIFO.

When the number of bytes in FIFO equals 15 minus FIFO-Threshold, the system bus is requested. This mechanism allows for a delay in bus acquisition equal to the number of empty bytes left in the Receive FIFO. If the bus is not acquired in time, a receive overrun occurs. The system bus is released when the Receive FIFO is emptied by the Receive DMA. Further acquisition of the bus starts when the Receive FIFO contains a number of bytes equal to 15 minus FIFO-Threshold. The mechanism is analogous to the Transmit FIFO.

NOTE:
In the case of a Receive FIFO overrun, the Channel Interface Module keeps overwriting the input buffer to the FIFO, and the bytes get lost. A special status will indicate this condition.

10.5 Bus Cycle Interleaving

The 82586 has four independent on-chip DMA channels:

- Receive DMA channel. Used for writing received frames to memory.
- Transmit DMA channel. Used for reading transmit frames from memory.
- CU input/output channel, used for CU read and write operations.
- RU input/output channel, used for RU read and write operations.

Receive and Transmit DMA channels operate in bursts controlled by the FIFO-Threshold mechanism. The CU and RU channels initiate single read or write operations as dictated by CU and RU activities. CU related activities are: CU Initialization, SCB processing including CA acceptance and SCB status update, CB execution, TBD prefetch. RU related activities are: CA acceptance, SCB status update, FD setup, RBD prefetch, end of receive frame processing. In the case that two or more channels require the system bus simultaneously, an arbitration mechanism located in the bus interface unit allocates the channels and determines priority between various requests.

TRANSMIT PROCESS

During Transmit operation, the CU prefetches the next Transmit Buffer Descriptor while the current data buffer is being read by the Transmit DMA channel. In order to minimize disturbances to the data rate, and to avoid a Transmit FIFO underrun, any two CU operations are separated by at least two data reads (if requested).

RECEIVE PROCESS

During reception, the RU prefetches the next Receive Buffer Descriptor while the current data buffer is being written by the Receive DMA channel. As in the case of transmit, any two RU operations are separated by at least two data writes. This rule minimizes disturbances to the data rate and helps to avoid Receive FIFO overruns.

SIMULTANEOUS RECEIVE AND TRANSMIT OPERATION

There are two cases where Transmit and Receive DMA channels work simultaneously.

In Internal/External Loopback configuration, both channels operate simultaneously, on an equal priority basis. Furthermore, CU or RU memory cycles are also interleaved with the same priority as Transmit and Receive. Effectively, if three sources compete for the bus, they operate on an interleaved basis.

The second case of simultaneous Transmit and Receive occurs when the receive frame arrives while the CU is setting up a transmission. The Transmit process continues simultaneously with Receive until the Transmit FIFO fills. As in Loopback, Transmit and Receive DMA priorities are equal. As soon as the receive frame ends, Transmit resumes operation.

The 82586 will interleave command related operations with receive related operations according to the rules described below.

- Control transfers (interaction with the System Control Block, Command Blocks, Receive Frame Descriptors and Buffer Descriptors) and data transfers (interaction with Data Buffers) are interleaved re-

gardless whether the transfers pertain to command or received operations. Two data transfers are executed for each control transfer.

- Data transfers to command and receive operations occur concurrently, on an alternating basis (one transfer in each direction).
- Control transfers related to receive operations have priority over command operations. In other words, command control transfers are suspended or delayed for as long as necessary to complete pending receive control transfers.
- Receive data transfers, command data transfers, and control transfers can occur concurrently. There will be one data transfer in each direction of each control transfer.

Consider the following example. Assume the 82586 is in the process of receiving a frame. The CPU wants to have a frame transmitted and activates Channel Attention.

While writing the received frame into system memory, the 82586 will concurrently read the appropriate command control information (SCB, Command Block, and TBD). If the 82586 needs to execute receive control transfers (beginning-of-frame processing, buffer lookahead, end-of-frame processing), it delays command control transfers. Two receive data transfers are executed for each (receive or command) control transfer.

When all control transfers related to the transmit command are completed, the 82586 will begin executing transmit data transfers. One transmit data transfer will be executed for each receive data transfer, a transmit data transfer and a receive control transfer (not necessarily in that sequence). Transmit data transfers will stop when the Transmit FIFO is filled up.

HOLDING THE BUS

In special cases, the 82586 does not release the bus, even though no specific reads or writes are executed. The purpose of this operation is to avoid bus acquisition and release dead times if several input/output operations are required. Another reason is to guarantee 82586 performance in critical timing cases.

The 82586 holds the bus in the following cases:

- RBD Prefetch. The bus is not released between two consecutive bus accesses to minimize prefetch time. This is important since minimum buffer length is strongly dependent on the time it takes the 82586 to prefetch a RBD.
- TBD Prefetch. Analogous to RBD Prefetch.
- Receive End of Frame Processing. The bus is not released throughout this process to ensure only one bus request.
- Transmit or Receive Buffer Switching. To minimize the danger of Transmit FIFO underruns or Receive FIFO overruns.

10.6 CPU/82586 (CA/INT) Handshake

The INT pin is used to notify the CPU about one or more of the following events:

- A Command Block with its 'I' bit set was executed (CX interrupt).
- A frame was received (FR interrupt).
- The CU became Not Active (CNA interrupt).
- The RU became Not Ready (RNR interrupt).

82586 INTERRUPT REQUEST SEQUENCE

Once an event requiring an interrupt has occurred, the following sequence is performed by the 82586:

1) INT pin is set to its low level (inactive).
2) The status word in SCB is written, denoting the source of the interrupt (CX, FR, CNA or RNR interrupt), together with the states of the CU and RU.
3) INT pin is raised (set to active).

82586 RESPONSE TO CA

The CU is responsible for control command acceptance, following the trailing edge on CA input. The CU will first finish all its higher priority activities and only then accept the control commands.

Higher priority CU activities that delay CA acceptance are:

a. Transmit Buffer Descriptor prefetch

b. Transmit buffer switching

c. Current Command Block completion

The 82586 will accept a CA prior to the setup of the next CB in the CBL.

The CU recognizes an RU control command and notifies the RU. The RU will first finish all its higher priority activities, and only then accept the control command.

Higher priority RU activities that delay CA acceptance are:

a. Receive Buffer Descriptor prefetch

b. Receive buffer switching

c. Receive end of frame processing

Only after the CU and RU have accepted the control command, the SCB command word is cleared. At that time, the CPU may issue the next CA to the 82586.

Internally to the 82586, the CA trailing edge is detected and latched. Prior to reading the SCB control command, the 82586 clears the latch. A new CA, given to the 82586 before the SCB command word is cleared, may be lost due to its being cleared before serviced. The user must refrain from such violations.

Upon detecting a falling edge on its CA input, the 82586 performs the CA acceptance sequence, as follows:

1) Determine which interrupt requests were acknowledged by the CPU. For each of them, clear the corresponding interrupt request bit in SCB status word.
2) Perform the control command acceptance procedure.
3) The INT pin is set LOW.
4) Write the SCB status word indicating the unacknowledged interrupt requests, and newly generated interrupt requests, together with CU and RU states.
5) If any interrupt request bit is active, set the INT pin to HIGH.

The 82586 does not wait for reception or transmission to end in order to process a CA. The SCB related operations will be carried out on an interleaved basis with the transmission or reception process.

11.0 NETWORK INTERFACE HARDWARE

The Network interface supports bit encoding, carrier sense, collision detection, link acquisition, and loopback.

NOTE:
The 82586 Receive-Byte-Machine and Receive-Bit-Machine are clocked by the Transmit Clock. Thus the Transmit Clock must be constantly applied to the 82586.

11.1 Encoding/Decoding

The 82586 receives an externally generated Transmit clock at the transmission bit rate. Data is transmitted in either NRZ (binary) or Manchester encoded form, depending on the chip configuration.

Due to the semi-static nature of the 82586's internal circuits, the Transmit clock HIGH time should not be longer than 1000 nanoseconds. Manchester encoded data requires 50% clock duty cycle. Therefore, when the 82586 is configured to perform Manchester encoding, the Transmit Clock frequency must be between 0.5 MHz and 10 MHz.

In the case of NRZ encoded data, Transmit Clock frequency can be from 100 KHz to 10 MHz, with the same limitation of the clock HIGH time as the Manchester encoded data. When Transmit data is idling, the TXD pin is held HIGH (logic '1').

The 82586 accepts an external Receive Clock, $\overline{\text{RXC}}$, that strobes the incoming Receive Data signal, RXD.

The Manchester/$\overline{\text{NRZ}}$ configuration parameter does not apply to the Receive data. The 82586 requires NRZ Receive data. Manchester data decoding is accomplished externally (the 82501 chip for IEEE 802.3). The Receive Clock can be presented to the 82586 in two ways:

- all the time
- only during the Receive frame

In the case of Receive Clock only during Receive frame, the $\overline{\text{RXC}}$ pin should be held inactive when no Receive frame exists.

Receive Clock frequencies can be from 100 KHz to 10 MHz with HIGH time not longer than 1000 nanoseconds.

11.2 Carrier Sense

Carrier Sense is an indication of activity on the link, i.e. a signal from a transmitting station has reached this station. The 82586 can be configured to either accept it externally, or generate Carrier Sense internally (for Serial Interface Devices that generate Receive clock only during actual reception). The internally generated Carrier Sense occurs when the Receive Clock is present.

When transmitting, 82586 behavior related to Carrier Sense is as follows:

- When ready to transmit, and if Carrier Sense is active, the 82586 defers.
- When $\overline{\text{CRS}}$ goes inactive ($\overline{\text{CRS}}$ is synchronized to the transmit clock), the 82586 sets its Interframe Spacing (IFS) timer.
- When the IFS timer expires, the 82586 initiates its transmission regardless of the state of $\overline{\text{CRS}}$.
- The 82586 will abort transmission if $\overline{\text{CRS}}$ goes inactive anytime after it transmits the preamble. An override option is available.

When receiving, the 82586's behavior related to $\overline{\text{CRS}}$ is as follows:

- The 82586 starts the IFS timer when $\overline{\text{CRS}}$ goes inactive. The receiver is insensitive to external signals during the time-out.
- Carrier Sense being active any time other than during the IFS time causes the 82586 to sample its Receive Data input at a rate determined by the Receive Clock.

- When receiving, the 82586 samples the data on the falling edge of the Receive Clock after $\overline{CRS}$ becomes active.

Carrier Sense signal going active edge can be asynchronous to both Receive and Transmit Clocks.

The 82586 requires five bit times from the instant the $\overline{CRS}$ pin goes active to achieve internal synchronization with the received bit stream. The sixth bit is the first bit sampled as data. In the End of Carrier frame delimiting method, the 82586 hunts for the SFD field (10101011). Conversely, if two consecutive '0' bits are detected before the SFD field, the frame is aborted.

To have a clean frame closure, $\overline{CRS}$ signal going inactive edge should be synchronized to Receive Clock.

Carrier Sense can be configured to undergo filtering to ignore negative going glitches. The filter is programmable to a number of bit times (0–7) that must be exceeded before the signal is considered valid. To be classified as a glitch, the glitch must meet set-up and hold times. $\overline{CRS}$ is synchronized to the Transmit Clock.

Figure 28A shows a negative going glitch on $\overline{CRS}$. If the glitch lasts shorter than the filter setting, it will be filtered, and the IFS timer will not start on the rising edge of $\overline{CRS}$.

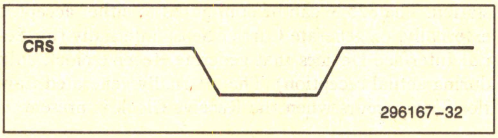


Figure 28A. The CRS Filter Stops Negative Going Glitches: the IFS Timer Does Not Become Active

Figure 28B shows a positive going pulse, which will not be filtered. At point 1, the $\overline{CRS}$ inactive will initiate the IFS timer and the 82586 will not respond to external signals for the IFS period. See section 12.3.

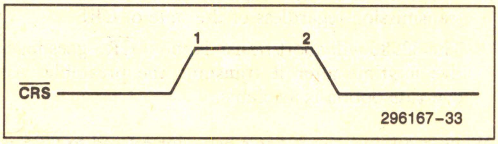


Figure 28B. The CRS Filter Passes Positive Going Glitches: the IFS Timer Becomes Active at Point 1

Positive going $\overline{CRS}$ glitches cause the 82586 back off timer to start; the 82586 will not respond to any external signal for the IFS period.

11.3 Collision Detection

Collision detection is usually done external to the 82586, typically by the Transceiver. An internal filtering mechanism prevents acceptance of a Collision Detection signal shorter than a configurable number of TCLK units (from 0 to 7).

The 82586 only looks for collision detect while actively transmitting, i.e. the first bit of the preamble has been transmitted. The collision is synchronized and recognized internally within a maximum of 4 Transmit Clock times after collision detection pin goes active.

The 82586 can be configured to cease transmission upon active Carrier Sense, instead of Collision Detect. This capability is called Internal Collision Detect. Internal Collision Detect is useful in two cases:

1) In conjunction with a class of transceiver that generates a Receive Data signal equal to the difference between the signal carried by the channel and the transmitted signal.
2) For point-to-point interconnection without using collision sensing transceivers. For example, collision detection could be realized by ANDing the $\overline{RTS}$ signal of each 82586, the result then tied to $\overline{CRS}$ of each 82586.

SQE TEST

Some transceivers (IEEE 802.3 compatible) generate the SignalQualityError Test, SQE TEST, (or 'heart beat') signal after the completion of each transmission to indicate proper operation of the collision detection circuitry. The SQE TEST signal is issued on the Collision Detect line, and is a 10 MHz signal, 10 ± 5 bit times in length. The 82586 checks the Collision Detect line for the SQE TEST for the duration of an Interframe Spacing after completion of transmission. The status of the next transmitted frame reports on the presence or absence of the SQE TEST. The SQE TEST mechanism is only meaningful if the 82586 is configured to external collision detect.

See section 12.3.

11.4 Serial Link Acquisition

The handshake between the 82586 and the Ethernet Serial Interface during transmission is accomplished with 'Request-to-Send' ($\overline{RTS}$) and 'Clear-to-Send' ($\overline{CTS}$) signals. $\overline{RTS}$ provides a means for turning on the Ethernet Serial Interface prior to actually sending bits. $\overline{CTS}$ is the means by which the Ethernet Serial Interface confirms that it is ready. It is also an external means for implementing a watchdog timer.

When the 82586 is ready to place bits on the serial link and $\overline{CRS}$ is inactive, it asserts $\overline{RTS}$ and awaits $\overline{CTS}$. Actual transmission starts within 1 transmit clock time after $\overline{CTS}$ rises. If it loses $\overline{CTS}$ prior to end of frame, transmission is aborted and error status is raised. An Ethernet Serial Interface that does not require this handshake may ground $\overline{CTS}$ (Intel 82501). The 82586 deactivates $\overline{RTS}$ after transmission is completed.

The 82586 can operate with Serial Interface Devices that either do or do not return Carrier Sense during transmission. The 82586 must be configured to one of the two cases. If configured to expect Carrier Sense, then transmission will stop if Carrier Sense goes inactive (and after the preamble). If configured to transmit on no $\overline{CRS}$, then the 82586 is indifferent to Carrier Sense going inactive.

11.5 Loopback

The Loopback Modes are called by setting the respective configuration bits.

If both bits are set, Internal Loopback Mode is valid, and overrides External Loopback. In Internal Loopback Mode, the 82586 is logically disconnected from the Serial Interface Unit, therefore, the 82586 does not monitor link activity. The 82586 connects the Transmit Data to the Receive Data Signal, and Transmit Clock to the Receive Clock. To avoid FIFO overruns and underruns, bit processing is performed at one quarter $\overline{TXC}$ frequency. The 82586 divides the Transmit Clock internally. The Receive Bit machine is clocked by $\overline{TXC}$. During Internal Loopback, NRZ data Encoding Mode is used regardless of the data encoding configuration bit. There is no limit to the number of data bytes looped back.

Ethernet Serial Interface and Transceiver diagnostics may be performed by configuring the 82586 to External Loopback. External Loopback Mode is performed at full link speed. Therefore, to avoid receive overruns, the frame length should be limited to a value that is a function of several parameters such as link speed, Preamble length and transceiver cable length. A frame of 18 bytes, including address type and CRC bytes, is guaranteed by chip design, not to cause overrun. See section 12.5.

In external loopback the 82586 can receive a frame from another station provided the 82586 detects carrier before it starts executing the Transmit command associated with loopback. The 82586 behaves as in the normal case of a pending transmit while a frame is being received.

The 82586 will only be able to receive a frame transmitted to itself (i.e. destination address = source address, or set to Promiscuous mode) in the loopback mode. Address Checking and Minimum Frame Size Checking are always performed by the 82586, even in loopback mode.

The Intel 82501, Ethernet Serial Interface, has a Loopback ($\overline{LPBCK}$) pin which if activated disconnects the 82501 from the Transceiver Link and transmitted data is fed back to the 82586. This mode enables diagnosing the 82501 without the Transceiver.

11.6 Interframe Spacing Timer

At the end of a transmission the Interframe Spacing Timer is triggered by the **later** of two events:

a. The last bit has been transmitted, or

b. Carrier has dropped.

This rule applies regardless of whether the entire frame was transmitted or the transmission was aborted due to a collision and the jam sent. In the latter case, the 82586 is sensitive to other stations not having completed their transmissions even after the 82586 has completed its jam; the 82586 defers until the channel goes 'not busy' and then sets the IFS timer.

This rule applies whether or not the 82586 is configured to expect carrier sense while transmitting; also, whether or not it is configured for internal collision detection as discussed in section 11.3.

12.0 CONFIGURATION PARAMETERS

The 82586 provides a high degree of flexibility via programmable parameters. This section summarizes the configuration parameters that may be modified (using the CONFIGURE command). Refer to section 8.4 for a summary of default settings.

12.1 Framing Parameters

PREAMBLE LENGTH (2 BITS)

Determines the length, in bytes, of the Preamble (including the SFD field).

00	2 bytes
01	4 bytes
10	8 bytes
11	16 bytes

For IEEE 802.3, these bits should be programmed to 10B (8 bytes). Preamble lengths other than 64 bits may

be tailored to particular networks. The preamble length is determined by the worst-case number of transceivers a frame passes before it reaches the destination.

ADDRESS LENGTH (3 BITS)

Determines the length, in bytes, of the address referred to by the 82586. This parameter applies to Individual, Source, Destination, Multicast, or Broadcast Addresses. An address length of 7 is treated as zero.

NOTE:
The Individual Address is set using the IA-SETUP command. Multicast addresses are set using the MC-SETUP command.

BROADCAST DISABLE (1 BIT)

Disables reception of frames with a Broadcast destination address or multicast addresses of 'all ones.' The Promiscuous Mode setting overrides the Broadcast disable.

0 Broadcast enabled

1 Broadcast disabled

CRC-16/CRC-32 (1 BIT)

Specifies which CRC polynomial is used for CRC generation and checking.

0 32-bit Autodin - II CRC

1 16-bit CCITT CRC

The 32-bit Autodin-II polynomial is $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$.

The 16-bit CCITT polynomial is $X^{16} + X^{12} + X^{5} + 1$.

NO CRC INSERTION (1 BIT)

Specifies whether or not the CRC is inserted after the information field during transmission. This capability allows CRC generation external to the 82586, enabling the user to verify the CRC checking mechanism during reception. The 82586 always checks for a valid CRC (CRC check cannot be disabled).

0 CRC is inserted

1 CRC is not inserted

BITSTUFFING/EOC (1 BIT)

Specifies the frame delineation method.

0 End of Carrier Framing (IEEE 802.3 compatible)

1 Bitstuffing Framing (HDLC like)

In the bitstuffing mode, the 82586 checks for the entire HDLC flag pattern (01111110), 7EH, before accepting it as a valid flag. Only one flag is allowed following the preamble bits. Two consecutive flags will be interpreted as a beginning and a closing flag to the 82586, i.e. a frame with no data.

The Bitstuffing/Flag technique offers high reliability, especially for networks susceptible to line ringing.

PADDING (1 BIT)

If padding is set, those frames shorter than the Slot Time will be padded with HDLC flags to the shortest frame that is longer than Slot Time (during transmission). Valid for Bitstuffing only. In End of Carrier framing, padding must be provided by user software.

0 Frame not padded

1 Frame padded (only in Bitstuffing)

MIN-FRAME-LENGTH (8 BITS)

Specifies the minimum frame size, in bytes. No frame that is shorter than the minimum will be accepted by the 82586. The 82586 does not accumulate statistics on short frames, and discards them if not configured to Save Bad Frames. NOTE: apart from this mechanism, there are other limitations on the minimum frame length:

First, frames that are shorter than 6 bytes (even in Save Bad Frame Mode, Promiscuous Mode, Address Length of Zero) are discarded. No status is reported on such received frames.

Second, for AL-LOC = 0 (when Address Control Location implies data separated from control), also frames shorter than 2 × ADDR-LEN + 2 (not including the Frame Check Sequence) are discarded.

For the IEEE 802.3 specification, this parameter should be set to 64. Transmission time of the 64 bytes ensure collision detection.

In slower or short topology networks, a shorter minimum frame size may be desirable to reduce channel overhead.

12.2 Link Management Parameters

INTERFRAME SPACING (8 BITS)

Specifies the time period (in TCLK units) that the 82586 must defer after detecting that Carrier Sense is inactive. The minimum value is 32, any value less than that defaults to 32.

SLOT TIME (11 BITS)

Specifies Slot Time for the Network (in TCLK units). Zero is interpreted as 2048. This value is used in calculating Backoff and Linear Priority delays. It must be longer than the maximum round trip time of a frame in the network plus jam time.

For IEEE 802.3, Slot Time should be set at 200H corresponding to 51.2 μs. However, it may be programmed to any number between 1 and 211.

The user may change the Slot Time to optimize the network to specific application environments. For many applications, like serial backplanes, this number will be significantly smaller than for IEEE 802.3. Setting the number to 7FFH allows the 82586 to operate over distances 8 times longer than specified in the IEEE 802.3 specification.

NUMBER OF RETRIES (4 BITS)

Specifies the maximum number of transmission retries (after a collision) that the 82586 performs before transmission is aborted by the 82586.

LINEAR PRIORITY (3 BITS)

Specifies the number of Slot Times periods the 82586 waits after Interframe Spacing or Backoff, before enabling transmission. A high number indicates low priority. Stations configured to zero, the highest priority, are equivalent to IEEE 802.3.

ACCELERATED CONTENTION RESOLUTION, ACR (3 BITS)

This parameter is added to the exponential number from which the random number is drawn for Backoff. In essence, ACR increases the range of random numbers to quickly resolve the case of stations contending for access to the network.

Specifically, let:

ACR - be the ACR priority number

N - be the number of collisions

r - be the random number multiplier of the slot time. r is a random number between 0 and 2 exp (min. [N + ACR, 10]).

ACR = 0, is equivalent to the IEEE 802.3 exponential backoff delay.

EXPONENTIAL BACKOFF METHOD (1 BIT)

This parameter determines when to start the back off time out:

0 Immediately after jamming or concurrent with Interframe Spacing. According to the IEEE 802.3 specification.

1 After deferring period expires (short topology).

This capability prevents inefficiencies and throughput loss in short topology or low data rate networks where the Interframe Spacing time may be longer than the slot time.

If the IEEE 802.3 backoff algorithm were applied to short topology networks where slot time is much smaller than Interframe Spacing, the 82586 would retry over and over again until the sum of the slot times exceeded the Interframe Spacing time, these retrys would waste channel bandwidth. See Figures 29A and 29B.

Linear Priority and Accelerated Contention Resolution can be combined in the short topology network environment. See Figure 30 (Alternate Backoff Method).

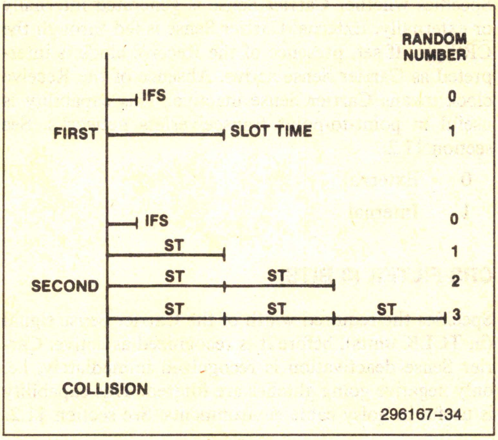


Figure 29A. IEEE 802.3 Retry Algorithm, Slot Time (ST) Greater Than IFS

12.3 Serial Interface Parameters

MANCHESTER/$\overline{\text{NRZ}}$ (1 BIT)

Specifies the bit encoding scheme used during transmission.

0 NRZ encoding (binary)

1 Manchester encoding

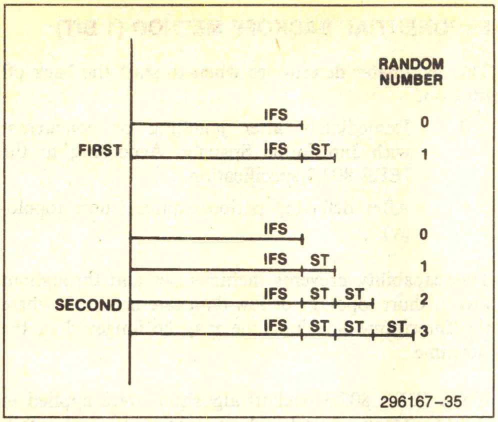


Figure 29B. Exponential Back Off Method = 1, Slot Time (ST) Less Than IFS

In the Manchester mode the 82586 requires external Receive clock recovery logic from the Receive data.

INTERNAL CRS (1 BIT)

Specifies whether Carrier Sense is generated internally or externally. External Carrier Sense is fed through the $\overline{CRS}$ pin. If set, presence of the Receive clock is interpreted as Carrier Sense active. Absence of the Receive clock means Carrier Sense inactive. This capability is useful in point-to-point transceiverless networks. See section 11.2.

0 External

1 Internal

CRS-FILTER (3 BITS)

Specifies the required width of the Carrier Sense signal (in TCLK units), before it is recognized as active. Carrier Sense deactivation is recognized immediately, i.e. only negative going glitches are filtered. This capability is useful in noisy cable environments. See section 11.2.

INTERNAL CDT (1 BIT)

Determines whether Collision Detect is generated internally or externally. External Collision Detect is fed through the CDT pin. Internal Collision Detect interprets presence of Carrier Sense during transmission as a collision. If internal Carrier Sense is used with internal Collision Detect, presence of the Receive clock during transmission will be interpreted as a collision.

0 External Collision Detect

1 Internal Collision Detect

See section 11.3.

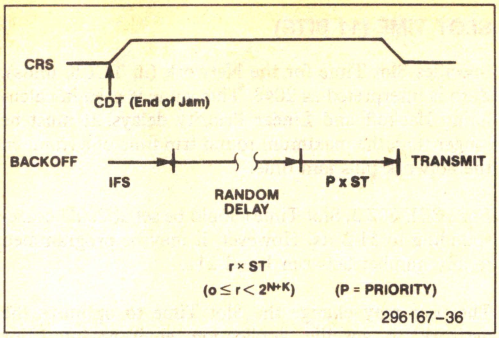


Figure 30. Combined Linear Priority and Accelerated Contention Resolution Using Alternate Backoff Method

CDT-FILTER (3 BITS)

Specifies the required width of the external Collision Detect signal (in TCLK units), before the 82586 recognizes that a collision occurred. Only negative going glitches are filtered.

TRANSMIT ON NO CRS (1 BIT)

Determines whether transmission, even if no Carrier Sense is returned from the Transceiver, is allowed. This option is used with Transceivers that do not return Carrier Sense during transmission.

0 Stop transmission if Carrier Sense drops

1 Transmit on no Carrier Sense

12.4 Host Interface Parameters

FIFO-THRESHOLD (4 BITS)

Specifies the point in the FIFO at which the 82586 requests the bus to transfer data to/from its internal FIFO from/to memory.

The FIFO limit is dependent upon the worst-case time from bus request to bus grant and serial channel and system clock rates. See section 10.4.

SRDY/$\overline{ARDY}$ (1 BIT)

Selects between Synchronous and Asynchronous Ready function of the SRDY/ARDY pin (active HI). If Asynchronous Ready is selected (SRDY/$\overline{ARDY}$ bit = 0), the SRDY/ARDY signal is synchronized internally by the 82586. If Synchronous Ready is selected (SRDY/$\overline{ARDY}$ bit = 1), the SRDY/ARDY signal is assumed to have been synchronized externally.

When using internal synchronization of the ready signal, one wait state may be added to each cycle, thus reducing the performance at most by 20%. Full performance systems must implement external ready synchronization. In that case, the SRDY/$\overline{\text{ARDY}}$ bit will be set to one and the external Ready signal will be connected to the SRDY/ARDY input pin.

0 Asynchronous Ready

1 Synchronous Ready

SAVE BAD FRAME (1 BIT)

Determines whether erroneous frames (CRC error, Alignment error, etc.) are to be discarded or saved. Erroneous frames are those where the OK bit in the Frame Descriptor status field is equal to zero. All frames are saved regardless of length.

In Save Bad Frame mode, the Frame Descriptor, as well as the Receive Buffer Descriptors and Receive Buffers are NOT reused for the next frame. In the complementary mode, all the descriptors and buffers used for bad frames will be reused, thus, not leaving any information about the lost frame except for updating the SCB statistical tallies.

0 Discard erroneous frames

1 Save erroneous frames

ADDRESS/LENGTH FIELD LOCATION (1 BIT)

Specifies the location of the Address and Length fields in the memory structures, and whether the Source Address is inserted during transmission.

0 Address and Length Fields are located in consecutive bytes in the Frame Descriptor. Source address is inserted by the 82586 during transmission.

1 The whole frame is located in the data buffers. Source Address is not inserted by the 82586.

This capability is useful in address/control schemes that the 82586 cannot manipulate. It is also helpful for diagnostics.

12.5 Network Management Parameters

INTERNAL LOOPBACK (1 BIT)

Specifies whether frames are internally looped back or not.

0 No internal loopback

1 Internal loopback

When set, the 82586 disconnects itself from the serial wire and logically connects TXD to RXD and $\overline{\text{TXC}}$ to $\overline{\text{RXC}}$. $\overline{\text{TXC}}$ must still be supplied by the user. Internally, $\overline{\text{TXC}}$ is divided by 4. This slows down the serial bit rate sufficiently to enable 82586 operation in full duplex. This will alter the effective values of all configure command parameters that are defined in terms of $\overline{\text{TxC}}$. Note that this is purely Internal Loopback capability. The INT-Loopback bit overrides the EXT-Loopback, i.e. having an INT-Loopback bit set, at the same time with EXT-Loopback, causes the 82586 to operate in Internal Loopback Mode. See section 11.5.

EXTERNAL LOOPBACK (1 BIT)

0 No external loopback

1 External loopback

The 82586 will receive and transmit simultaneously, at full rate, a frame limited to 18 bytes (including the Frame Check Sequence). This allows checking of external hardware as well as the serial link to the transceiver. For IEEE 802.3 transceivers, since the transmitted data is fed back via the receive pair, practically nothing has to be done to perform External Loopback. For other transceiver types, the user is responsible for external transmit-receive interconnection. See section 11.5.

NOTE:

Internal Loopback bit overrides External Loopback bit.

PROMISCUOUS MODE (1 BIT)

Determines whether or not the 82586 accepts all frames regardless of their Destination Address.

0 Normal address filtering

1 Promiscuous mode

When Promiscuous mode is set, the optional broadcast disable is overridden.

13.0 INTERNAL ARCHITECTURE

The 82586 is divided into three functional modules: Host Interface Module, Channel Interface Module, and FIFO Module. The Host Interface Module communicates with the host CPU and shared memory via the Bus Interface. It performs direct memory access, buffer chaining, and interpretation of high level commands.

The Channel Interface Module communicates with the Network via the Network Interface. It performs network related activities: framing, link management, address filtering, data encoding, and network management.

The two units inter-communicate via the FIFO Module that consists of two 16-byte FIFOs.

A block diagram of the 82586 is shown in Figure 31. The three modules are separated by dashed lines.

13.1 The Host Interface Module

The Host Interface Module appears on the right side of Figure 31. It consists of separate units for data, address, and control interfacing to the local bus, all under control of two micro-sequencers (for executing commands and receiving frames, respectively), a micro-instruction ROM, an Arithmetic Logic Unit, and a register file.

Communication between units of the Host Interface Module is by means of an internal 16-bit bus (P-bus), controlled by the two micro-sequencers: the Command and Receive Units. Both sequencers operate on micro-instructions from a common ROM; each is dedicated to a separate class of tasks, and has its own program counter and stack. Only one of them can run at any given moment.

The Command Unit fetches commands from shared memory space, and controls other units of the Host Interface Module as necessary for executing the commands. It also receives status signals, processes them, and updates shared memory as required. The Command Unit controls the DMA machine, loads starting pointers and byte counts for DMA transfers, triggers the start of transfers, and aborts them if necessary. It uses the Arithmetic Logic Unit (ALU) as needed. Its commands are sent to the Channel Interface Module via the Transmit FIFO. The Command Unit responds to the Channel Attention (CA) signal from the host CPU, and manages the initialization process.

The Receive Unit fetches, from shared memory, information defining buffer availability, size, and location, and controls data transfer to the buffers. In executing these tasks, it performs similar functions as described above for the Command Unit, with the exception of initialization and response to Channel Attention.

The Micro-Instruction ROM supplies micro-program to the Command and Receive Units. It contains a thousand words, 20-bit long. The Arithmetic Logic Unit is used by the micro sequencer to perform simple arithmetic and logical operations. The Register File consists of twenty four 16-bit registers, plus an additional 48 flags. The Register File makes up an internal data storage facility for the Micro-Machine.

The DMA Unit is a memory address generator which operates on starting addresses and byte counts supplied by the Micro-sequencers to transfer information between the 82586 and shared memory. The unit is composed of four channels. Two channels are designed for block transfers, one each for transmission and reception; each of these includes a 14-bit byte counter and a

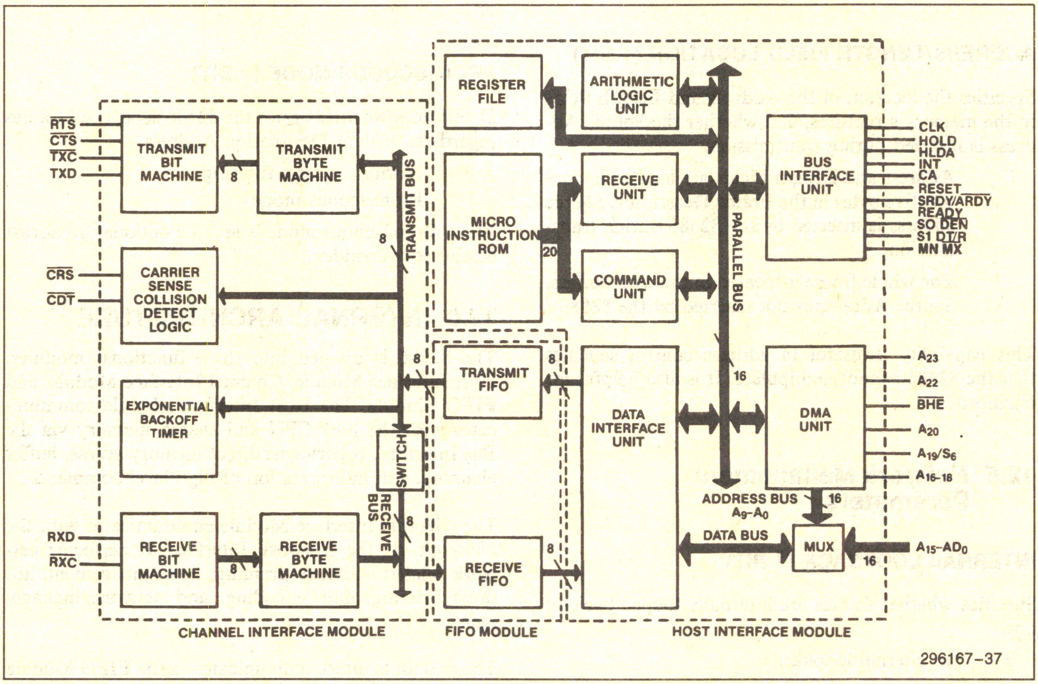


Figure 31. Block Diagram of the 82586

24-bit absolute address register. The remaining two channels are for single word transfers, one for each of the micro sequencers; each includes a 16-bit address offset register. The two channels share a common 24-bit base address register.

The sixteen lower order lines of the generated addresses are multiplexed with the data lines in Word mode.

All control signals between the 82586 and the local bus are generated or received by the Bus Interface Unit (BIU). The BIU also handles the CA line, used by the host CPU for the 82586 attention calls, and the INT line, for the 82586 to request the attention of the host. CLK is a clock supplied by the host, and is used for all Host Interface Module and FIFOs timing. Signals that control the local bus are also defined in terms of CLK. These signals are optimized for the iAPX 186, 8086, and 8088 bus.

The Data Interface Unit is a switching matrix which routes data and status signals to the appropriate destinations by interconnecting input and output as required. Command data are routed from the P-bus to the Transmit FIFO; transmit and receive status signals are sent from the Receive FIFO to the external bus; and data to be transmitted are routed from the external bus to the Transmit FIFO. Note that the two FIFOs are 8 bits wide, while the P-bus width is 16 bits. Data packing and unpacking are performed in the Data Interface Unit as required.

13.2 The Channel Interface Module

This is the second portion of the 82586, shown on the left side in Figure 31. It contains separate units that perform transmission and reception over the Network Interface.

The Transmission logic is composed of three units: Transmit Byte Machine, Transmit Bit Machine and Carrier Sense/Collision Detect Logic. The Transmit Byte Machine interprets commands from the Transmit FIFO, executes them, and generates appropriate status signals, which are returned through the Receive FIFO.

During transmission, the Transmit-Byte-Machine assembles data frames, calculates and appends the Frame Check Sequence, and passes the frames to the Transmit Bit Machine. The latter converts the frame into a pulse train and transmits it to the Network Interface. The Transmit Bit Machine performs Bitstuffing (if configured to do so).

The Exponential Backoff timer implements Backoff, Defer, Wait and Priority algorithms and other 82586 timers.

The Receive logic consists of the Receive Byte Machine and Receive Bit Machine. When a frame is received from the Network, the Receive Bit Machine strips the Preamble and SFD Field from the frame, and determines the end of the received frame by the occurrence of either EOF flag or End of Carrier. If Bitstuffing is active, inserted zeros are deleted, and computes and verifies the CRC (Cyclic Redundancy Check).

The Receive-Byte-Machine checks the Destination Address of the received frame to determine if the Individual Address matches the Receiver's; for Multicast transmissions, it performs the hash filter function to determine whether the frame is directed to this Station.

Received data and status signals are relayed through the Receive FIFO to the Host Interface Module.

13.3 The FIFO Module

The 82586 contains two 16-byte FIFO storage arrays located between the Host Interface Module and the Channel Interface Module. One FIFO transfers data in the transmit direction and the other in the receive direction.

The FIFOs improve local bus utilization by virtue of temporary data storage on the way to or from the Network. For continuous transmission in the absence of the transmit FIFO, the local bus would have to be dedicated to frame transfer during the entire transmission and the host CPU would be unable to use the bus for other tasks. The FIFO allows the 82586 to relinquish the local bus for intermittent intervals during transmission. During these intervals data in the FIFO empties toward the Network, maintaining transmission until the bus returns to the 82586 control.

Similarly, the receive FIFO accumulates incoming bytes while the bus is otherwise occcupied, reception continues without data loss.

CHAPTER 3
PROGRAMMING THE 82586

INTRODUCTION

The 82586 LAN Coprocessor handles most of the functions of the data link and physical link layers of the ISO Open Systems Interconnect model. It does this with a minimum amount of supervision by the host CPU and usually executes concurrently with the host CPU. The host CPU and the 82586 communicate through a set of shared memory control structures. The 82586 is a DMA master which allows it to operate on these control structures as needed to handle commands and manage its own buffers.

The asynchronous nature of data communications and the autonomous operation of the 82586 requires special attention to the CPU/82586 software interface. In order to simplify communication between the CPU and the 82586, a protocol has been defined that must be used by both units. This chapter discusses the algorithms that the CPU must use in communicating with 82586. Since the 82586 is expecting the CPU to follow these algorithms, the user must be sure that the CPU does so. Failure to follow the algorithms may result in system failure.

This chapter will discuss two basic issues. The first is how the 82586 can be placed into a variety of operating system environments. This topic is important because the 82586 is an intelligent peripheral component: a coprocessor. The second issue is the algorithms by which the host CPU controls the 82586, that is the details of device control. To properly use the 82586, the user must address both sets of issues.

1.0 FITTING THE 82586 INTO A SYSTEM

The first problem to be solved in using the 82586 is to define how the 82586 fits into the software environment of a host system. The 82586 is more powerful than most peripheral components and may present some unique problems fitting it into a system. The 82586 is more a coprocessor than it is a peripheral component and must be treated as such. The 82586 is a coprocessor primarily due to its buffer management capabilities as well as its ability to chain commands.

In order to understand how the 82586 might fit into a system it is helpful to consider a general model of how distributed systems software might be implemented, see Figure 1.

Figure 1 shows four distinct entities:

1) The 82586 itself

2) The 82586 handler

3) Upper Layer Communications Software (ULCS)

4) User application software.

The 82586 is simply the controller itself. Inputs from the host CPU include Commands, Command Blocks, FDs, and Receive buffers as well as Channel Attention signals. Outputs from the 82586 to the host include completed commands, received frames and interrupts.

The 82586 handler consists of software routines that actually control the 82586. These routines perform:

- Generating Channel Attentions to the 82586
- Receiving interrupts from the 82586
- Reading and writing the 82586 control structures
- Giving commands to the 82586 and determining when they are completed
- providing free (unused) buffers to the 82586 and determining when they have been filled.

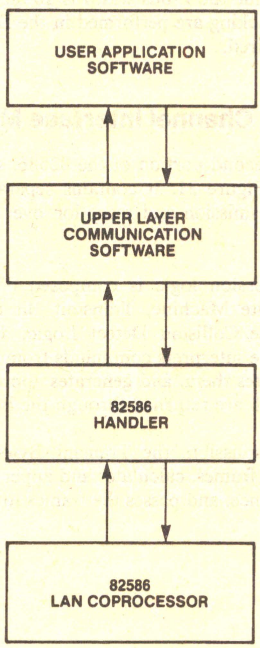


Figure 1. Distributed System Software Model

The ULCS consists of software that provides network capability for user application software. The ULCS usually consists of Transport Layer (Layer Four of the OSI model) software that provides reliable, process to process delivery service. Such software is required in one form or another because the CSMA/CD protocols supported by the 82586 (such as Ethernet and IEEE 802.3) provide only datagram or 'best effort' box to box delivery service. The Transport layer provides enhanced communication services compared to datagram based data links. These services include process-to-process message delivery. This capability is useful since there are usually several sources, or sinks, of data at a single node. Also provided is guaranteed end-to-end message service. Transport ensures that data is not lost, duplicated, or delivered out of order. The ULCS interfaces to the 82586 handler when it wishes to transmit a frame or when the handler pases a received frame to it.

The User Application Software consists of software that uses the communications channel to perform tasks in a distributed manner. This software uses the ULCS to provide reliable communication path to other processes on the network. This software is usually not aware of the presence of the 82586.

The following sections address the problem of how the 82586 handler fits into a system. This problem involves two major areas of discussion. The first concerns the nature of the interface between the 82586 handler, the host operating system and the ULCS. That is, how does the 82586 handler fit into the operating system environment. Before the handler for the 82586 can be written, there should be clear model in mind for how it will fit into the system. The second major area concerns buffer management. That is, how buffers are allocated, utilized and moved between the two entities. Understanding these issues is the topic of the following sections.

2.0 THE 82586 HANDLER

There are basically two approaches for fitting the 82586 handler into an operating system. The first has the 82586 handler appear as a standard I/O driver. The second makes the 82586 a special type of device that bypasses normal I/O conventions. In the latter model, the 82586 handler would likely be integrated into the ULCS, see Figure 2.

The basic issue in the two approaches is to what extent will the 82586 be allowed to control the rest of the system, especially the communications software, in order to allow full use of its capabilities. If the 82586 must be fitted into existing structures, it is unlikely that its full power can be utilized. Alternatively, to extract the full benefits of the 82586, the designer will have to stretch the operating system to accommodate the 82586.

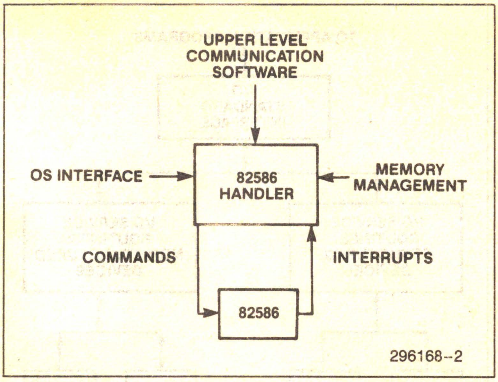


Figure 2. The 82586 Integrated Into ULCS

2.1 The 82586 Handler as a Standard Device Driver

Most modern operating systems have an I/O system that can be broken down into three parts (Figure 3). The first part consists of a standard interface used by all application programs. This interface is used by all types of devices ranging from disks to line printers. Its purpose is to make all devices look alike so that application programs can operate in a device independent manner. The second part consists of I/O service routines that handle particular types of devices. Usually two major types of devices are considered, structured devices such as disk or tape files and nonstructured devices such as TTY drivers and line printers. The third part are the device drivers which are routines that actually manipulate the hardware such as the disk controller or USART.

Device driver interfaces are usually standardized within an operating system to allow varieties of devices to be used without rewriting any of the I/O service routines. The interface is usually fairly simple because its goal is to allow as many types of devices to be used as possible.

In this type of environment the 82586 handler would fit into the system as a device driver for an unstructured device. The user would issue commands to the 82586 via the standard I/O interface which would be relayed to its device driver (Figure 4). In such an environment the ULCS would exist as an application software routine using a standard interface to the 82586 handler.

There are a number of advantages using this approach. First, it may be the only way to fit the handler into the system. If the system has memory protection or virtual memory, being part of the I/O system may be the only

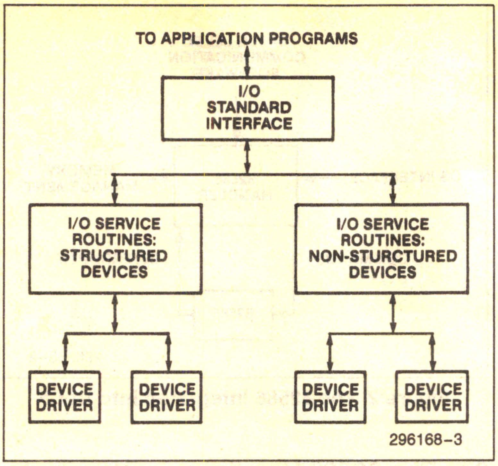


Figure 3. Common I/O System Structure

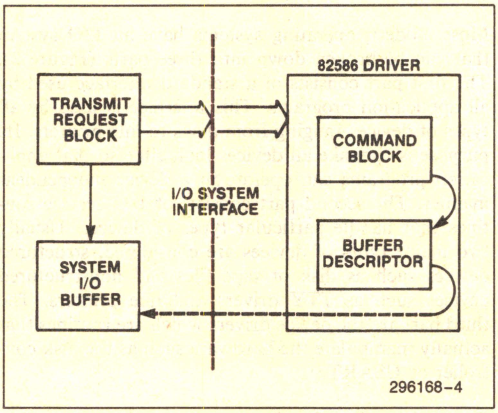


Figure 4. The 82586 Handler as a Standard Device Driver

way the handler can get access to user memory or actual physical memory (rather than virtual memory). Second, there would be a reasonable degree of device independence. The ULCS could be written independent of the 82586 and so other data links could be inserted with minimum changes to the ULCS. Third, the 82586 handler would not have to reinvent the mechanisms already used by all device drivers, such as interrupt handling, buffer management or fault handling.

However, this approach is likely to prevent many of the high level function capabilities of the 82586 from being used. The standard I/O interface and the device driver interfaces are likely to prevent use of buffer and command block chaining. As noted above, these standard interfaces are usually fairly general, and especially at the device driver level, assume rather simple devices (i.e. devices that perform one task at a time, do not manage their own memory and require constant attention from the CPU). In addition, the buffers that are passed across such interfaces are usually assumed to be one large contiguous block of memory. It could be fairly difficult trying to fit the 82586's linked list mechanisms into such a model. In some operating systems, only one command at a time can be issued across the interface. Often such an interface is not very good for asynchronous events (increased probability of losing frames). In addition, the performance across such an interface may be slow. Despite these problems, undoubtedly there are many systems in which this is the best approach. In such situations there are a number of mechanisms that allow many of the features of the 82586 to be used.

Looking first at command block handling, a common model of operation would be one in which an operating system command block with something like a WRITE command contained inside it would be passed to the device driver. There would also be a pointer in this command block to an area of memory containing data to be 'written' (a System I/O Buffer). The 82586 device driver could interpret this WRITE command as being a TRANSMIT command with the data being the message to be sent. Inside the device driver would be an 82586 Command Block (CB) and a Transmit Buffer Descriptor (TBD). When the driver receives the operating system request block it will set up its internal CB to the desired function and its TBD to point to the System I/O Buffer. The driver then gives the CB and TBD to the 82586 for execution. When the 82586 has completed processing, the Driver returns completed operating system command block back to the operating system. The internal CB and TBD are then freed.

There are a variety of variations on this technique but all are based upon an internal CB/TBD within the device driver being used to pass the command to the 82586. For example, one might consider passing the 82586 command block and message as part of the data to 'written'. In this case the contents of the provided 82586 Command Block would be copied to the internal CB and the TBD set up to the message part of the data. By having an internal CB and TBD the need for them to be in the same 64K byte segment as FDs, RBDs, and the SCB can be easily met.

Turning to receive handling there are a variety of options. The most simple approach is to reserve a FD, RBD, and single buffer (long enough to contain the biggest message that will be received) within the device driver. When the 82586 receives a frame, it fills the buffer and passes it back to the driver. The operating system would recover the frame by issuing a READ with a System I/O Buffer. The received frame would be copied into the System I/O Buffer and the operating system READ command returned. While very simple

this approach suffers from possible poor performance due to lost messages. Lost messages are caused by the single receive buffer within the device driver being freed only when the operating system posts a receive buffer. There is also the cost of doing the copy into the System I/O Buffer.

A variation on this technique that addresses the copy problem is to move the data directly into the System I/O Buffer by using the buffers supplied by the READ command. As in the command block case, internal FDs and RBDs would be used as control structures. With either approach a frame can get lost if the System I/O Buffer is not supplied with new buffers frequently.

An alternative approach is to reserve a circular list of FDs, RBDs within the device driver, see Figure 5. The 82586 places received frames into the buffers. When the operating system issues a READ command, the 82586 driver copies this information into the supplied System I/O Buffer. The device driver reclaims the old FD, RBD, and buffers.

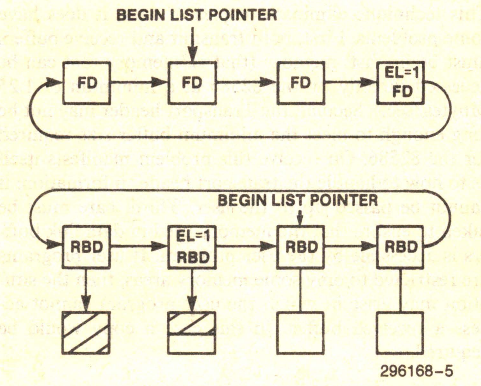


Figure 5. 82586 Device Driver Using Circular Lists

The circular list is managed using a FIFO philosophy. Frames are always processed in the order they were received, and processed FDs and RBDs are replaced in order. The links among the FDs on the RDL and the RBDs on the FBL are never altered. The EL bit is used to mark the logical end of the list. The 82586 itself manages the linking FDs to RBDs as they are needed. If the 82586 runs out of FDs or RBDs, it enters the No Resources state. In order to know the logical beginning of the lists, the devide driver will maintain a pointer to the 'first' free FD and RBD. As frames are received The driver will always know which FD is to be used next. As FDs and RBDs move from the empty to filled state, the pointers should be moved to the next FD and RBD. Recovering FDs and RBDs simply involves changing the EL bit from one to zero on the previous FD and RBDs to the one being recovered.

This circular list approach takes better advantage of the 82586's capabilities in that the 82586 can receive frames asynchronously to the operating system reading them. Also, the 82586's buffer chaining capability can be used to obtain more efficient memory usage.

2.2 The 82586 Handler as a Special Driver

If the 82586 handler is treated as a standard device driver, device independence is maintained, but at the cost of performance. Performance is degraded for two reasons. First, copies of data must be made between the System I/O Buffer and the driver. Second, frames may be lost due to a lack of buffers. This suggests that if possible, another approach be found.

This other approach consists of moving the 82586 handler outside of the I/O system and be treated as a special kind of device. In this model, the 82586 handler would exist as a separate entity, most likely made part of the ULCS routines, see Figure 6. The only support required from the operating system would be interrupt handling. The ULCS would likely operate on linked buffer structures and perhaps be aware of the control structures of the 82586. The 82586 handler would still manipulate the control blocks given it by the ULCS and interface to the 82586, but operate much closer to the ULCS. In this environement more attention must be paid to the buffer management model used by the 82586 and the ULCS. The 82586 can operate with a variety of buffer models in this type of environment although there is one that can be considered its 'design model.' This model and several others will be discussed in the following section.

The 'Design' memory management model for the 82586 is shown in Figure 7. This model assumes an explicit Transport (and perhaps Network) Layer. Two pools of memory, one for Transmit (called the CB pool), and the other for Receive (called the RFA pool) contain various control blocks needed for 82586 operation. The CB pool contains CBs, TBDs, and transmit buffers. The RFA pool contains FDs, RBDs, and receiver buffers. Blocks do not move between pools. These data structures are 'owned' by the 82586 handler. Usually these pools are independent of the operating system's Free Space Manager (it is possible for them to be part of a system free space manager, but this will significantly increase overhead). Both pools are 'owned' by the 82586 handler or by the 82586 itself. The CB pool belongs to the handler while the RFA pool belongs to the 82586 itself. The RFA pool is basically the memory pool managed by the 82586 Receive Unit.

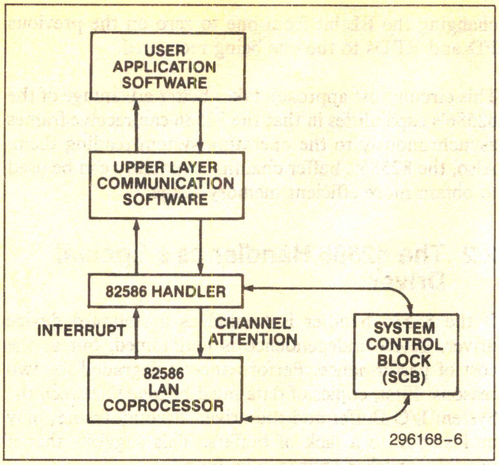


Figure 6. The 82586 Handler as Part of the Communications Software

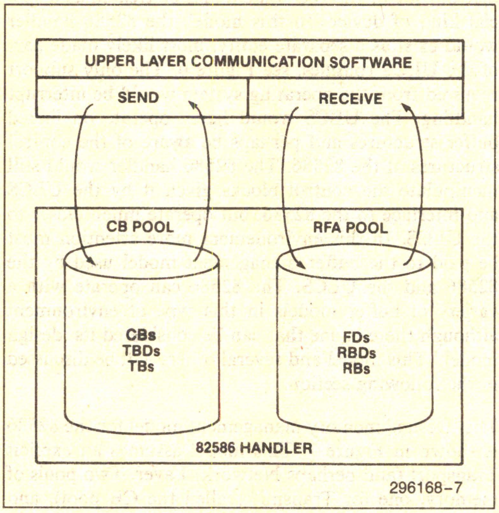


Figure 7. 82586 Handler Memory Management Model

To transmit a Frame, the Transport Layer requests a buffer from the 82586 handler. This might be done by making a subroutine call to the handler. The handler returns a command block and enough buffers (with TBDs) from the CB pool to the Transport Layer. The Transport Layer will fill the buffer with some portion of the user message and initialize the CB to whatever values are required for the operation desired. The CB (and buffer) are then returned to the handler where it is given to the 82586. When the command is executed by the 82586, the handler will receive an interrupt and will reclaim the CB and TBD and places them back into the pool.

On the receive side, the RFA pool is managed by the 82586 ifself. When a frame has been received, the 82586 interrupts the handler. The handler passes the FD, RBD(s), and receive buffer(s) to the receive side of the Transport Layer. The Transport Layer extracts what it needs, and returns the FD, RBD(s), and receive buffers(s) to the handler. The handler reclaims these structures and places them at the end of the Receive Frame Area managed by the 82586.

As noted above there are a variety of memory management models that the 82586 can operate with besides the 'design' model. Another model of interest eliminates the copy of data from user buffers into transmit data link buffers and from receive data link buffers to user buffers. One approach to eliminate copies of data is to eliminate the transmit buffers from the CB pool and only provide CBs and TBDs. The Transport Layer software then sets up pointers to header information and user data using the TBDs. On receive, the Transport Layer passes on the data link buffers to the user and returns only the FDs and RBDs (plus new receive buffers to replace those passed to the user).

This technique eliminates copies of data, it does have some problems. First, both transmit and receive buffers must be in 'fast' memory (that is, memory that can be accessed directly by the 82586 at a minimum of 1.25 Mbytes/sec.). Second, the Transport header may not be long enough to meet the minimum buffer size required for the 82586. On receive this problem manifests itself as to how to handle the transport header information; it cannot be passed up to the user. Third, care must be taken to ensure that the memory used in data link buffers is accessible by the user program. If user programs are restricted to only some memory areas, then the situation may arise in which the user program cannot access a received buffer. In this case a copy would be required.

As noted above the 82586 was designed to work with one buffer management model. The 82586 does operate with other models provided the algorithms in the following sections are followed. **It is important to recognize that there must be a buffer management model present in the system. It should be a model that is chosen upfront in the system design, not after.**

3.0 INITIALIZATION OF THE 82586

Initialization of the 82586 occurs in these three phases. The first phase involves use of the SCP and the ISCP to locate the SCB and define the bus width. The second phase involves issuing Configure, Individual Address Setup and Multicast Address Setup commands. The third phase involves starting the RU by supplying it with an initial allocation of FDs and RBDs. This section addresses only the first two phases; section 6.0 addresses starting the RU. It should be noted that it is

important that three phases be done in the order indicated. Starting the RU prior to setting up the individual address is generally undesireable.

The SCP and ISCP are designed for maximum flexibility in locating the SCB. Usually in iAPX 86 family systems, the SCP is located in ROM together with the processor bootstrap routines. The model of operation is that the SCP will point to the ISCP which is located in RAM. The address of the SCB can be written into this RAM location so that it can vary with different system configurations. The SCP and ISCP will exist only for initialization purposes. Once the SCB is located, the SCP and ISCP locations can be reclaimed. The SCB should be initialized prior to beginning the SCP/ISCP sequence.

The second phase involves issuing commands that configure the 82586. If the 82586 is used in IEEE 802.3 configuration, it is not necessary to issue a configure command since the 82586 initializes itself as an IEEE 802.3 controller. Otherwise the user must issue the Configure command. After configuration, the user should issue an Individual Address Setup command to load the 82586 with the host address. This procedure may be followed by the MC Setup command as required. These commands may be chained together or issued one at a time.

After address setup, the RU can be started and supplied with FDs and RBDs, see section 6.4.

4.0 SIMPLE COMMAND PROCESSING

With an understanding of how the 82586 handler fits into a system, the operations to be performed by the handler can be discussed. This section discusses giving commands to the 82586 and servicing 82586 interrupts. Simple command processing in this context means that the 82586 handler accepts only one command at a time for the 82586. ULCS is not allowed to give a second command to the handler before the previous command has been executed. More complex processing will be discussed in section 5.0.

4.1 Adding CBs to the CBL

Simple command processing means that commands are issued one at a time to the 82856. The handler is given a single CB by the ULCS, writes the address of the CB into the CBL_OFFSET field in the SCB, places a Start CU command into the SCB, and issues a CA. The 82586 will accept the Start CU command and clear the SCB field. It will then execute the CB, update status in the SCB and generate an interrupt to the handler. The handler will process the interrupt and then reclaim the CB for future use. This procedure can be repeated as required.

Figure 8 illustrates this basic procedure of giving a new CB to the 82586. **Note that the CPU must mask off interrupts from the 82586 during various parts of the algorithm.** Masking interrupts will prevent the interrupt service routine from issuing commands to the 82586 which might overwrite the command just given to the 82586 by the command processing routine. Overwriting might occur if an interrupt occurred between the recognition that the SCB command field was clear and the issuing of the CA. Either the command processing routine or the interrupt service routine could find their command to the 82586 being overwritten by the other. The result of this race condition would likely be a system hang up.

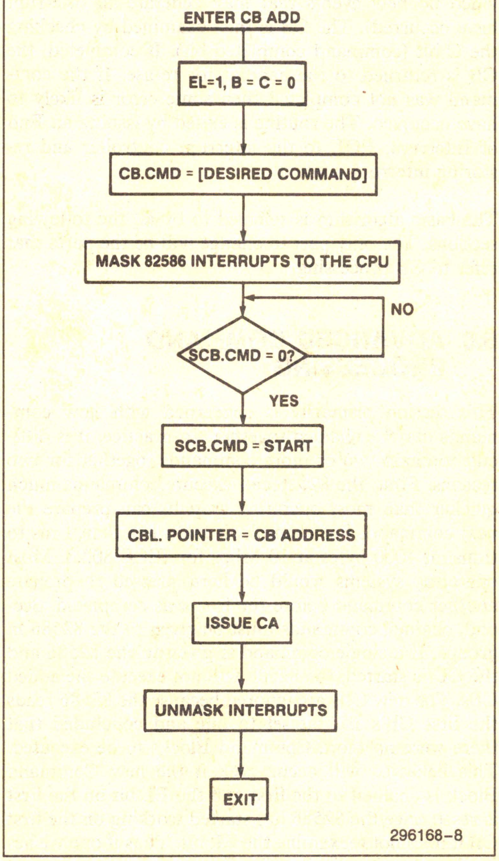


Figure 8. Simple Procedure to Add a New CB to the CBL

4.2 Basic Interrupt Service Routine

When a command has been completed by the 82586, an interrupt will be issued to the host CPU (if the I bit was

set in the (CB). Figure 9 shows the basic interrupt service routine. This routine's interrupt acknowledgement sequence will be used over again in the command and receive interrupt service routines that will be described later. The routine starts by saving CPU context (this may have already been done by the operating system). The handler then waits for the SCB command word to clear, which signifies that the 82586 has no pending control commands. The handler determines the nature of the interrupt, sets the appropriate ACK bits in the SCB Command Word, and issues a Channel Attention (CA) to mark acknowledgement of the interrupt. The handler then waits for the 82586 to accept confirmation of the interrupt acknowledgement by clearing the SCB command field and removing the interrupt signal (provided no new events that may generate an interrupt have occurred). The CB is then examined by checking the C bit (command completed bit). If completed, the CB is returned to the handler for re-use. If the command was not completed then some error is likely to have occurred. The routine is exited by issuing an End of Interrupt, EOI, to the interrupt controller and restoring interrupt context.

The basic algorithm is referred to by all the following sections. The only part to change will be the parts that refer to CB processing.

5.0 ADVANCED COMMAND PROCESSING

This section primarily is concerned with how commands may be chained together. In practice, it is difficult to chain two or more commands together for two reasons. First, the 82586 can execute commands much quicker than most operating systems can prepare the next command. For example, it takes less than 1 ms to transmit 1000 bytes at 10 Mbps for IEEE 802.3. Most operating systems would be hard pressed to prepare another command before the first was completed. Second, chained commands must be given to the 82586 in groups. If a single command is given to the 82586 and the CU is started, the 82586 will not execute the added CBs. The new CBs are ignored because the 82586 reads the first CB's EL bit set to one and concluded that there were no more Command Blocks to be executed. This behavior will occur even if the new Command Block is chained to the first and the EL bit on the first is reset; once the 82586 has started working on the first CB it does not reexamine the EL bit. Thus if one wishes to chain commands it will be necessary to ensure that the 82586 has at least one unfetched Command Block present. **This rule requires that the 82586 be started with at least two Command Blocks present.**

Despite these problems there are situations in which some form of Command Block chaining might be useful. The first case is where the ULCS may have the capability to occasionally give the 82586 handler multi-

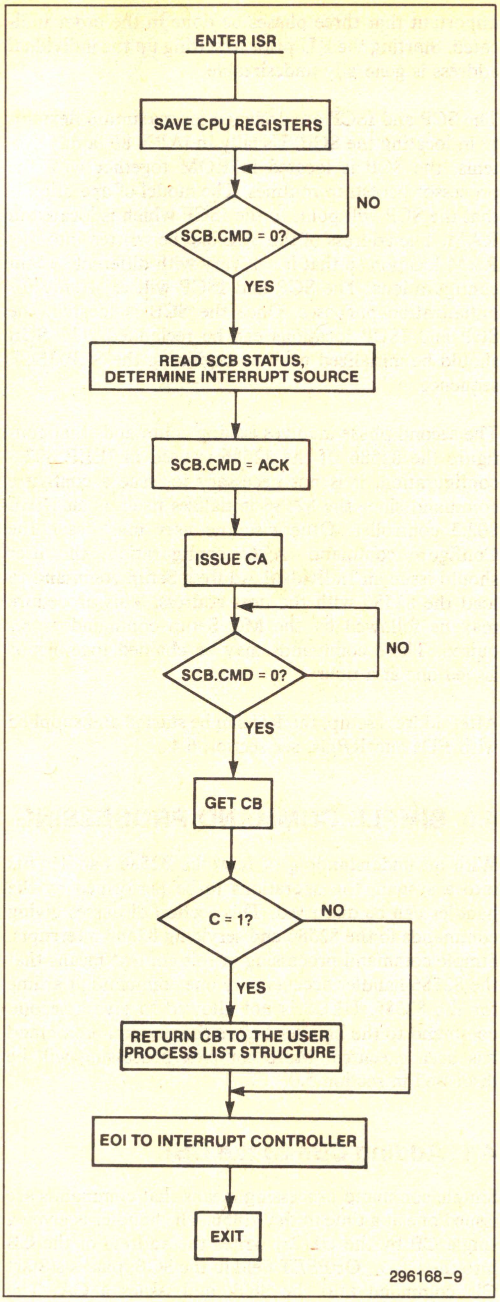


Figure 9. Basic Interrupt Service Routine Sequence

ple Command Blocks (not at the same time) and some structure is desired to handle it. In this case the commands are most likely to be executed one at a time by the 82586 simply because they are given to the handler one at a time. This case is called **static lists**. In this approach Command Blocks are chained together on a list but only one Command Block is actually given to the 82586 at a time. Thus the 82586 will execute one Command Block, halt (the CU goes to the IDLE state), and interrupt the host CPU. The handler would then process the completed Command Block and, if another one is on the list, give it to the 82586 for execution. If there are no commands, the CU is not restarted. Given how fast most CPUs can actually prepare request blocks and the manner in which the 82586 operates upon lists, static lists are probably the best overall approach.

The second approach is similar to the first except that the 82586 can be given multiple commands to execute. This case, called **dynamic lists**, occurs only if there are two or more unexecuted commands on the Command List and the CU is in the IDLE state (requiring it to be STARTed). If there is only one command on the list (or the 82586 is on the last Command Block of a list of commands) then trying to add additional commands yields nothing. Dynamic lists are an attempt to take advantage of the fact that the 82586 executes one command at a time and does not look beyond the current CB being executed (except for the implications of setting the EL bit). Most of the time, dynamic lists will function as static lists, but occasionally an interrupt may be saved.

This section discusses both static and dynamic lists. They share a very similar method of adding Command Blocks to the end of the CBL. The major difference is in the interrupt service routine. As a result, the algorithm for chaining commands will cover both cases, and interrupt processing will be discussed separately.

5.1 Adding Command Blocks to Static and Dynamic Lists

To add a CB to the list, the ULCS will most likely call a procedure that does it. A procedure for CB chaining for static and dynamic lists is shown in Figure 10. The handler defines two pointers: Begin.CBL and End.CBL. Begin.CBL locates the first CB on the list while End.CBL locates the last CB. If there are no CBs on the list then Begin.CBL is set to 0FFFFH. The pointers bound the list of CBs as defined by the CPU. The 82586 will operate on all or some part of the CBs on the list.

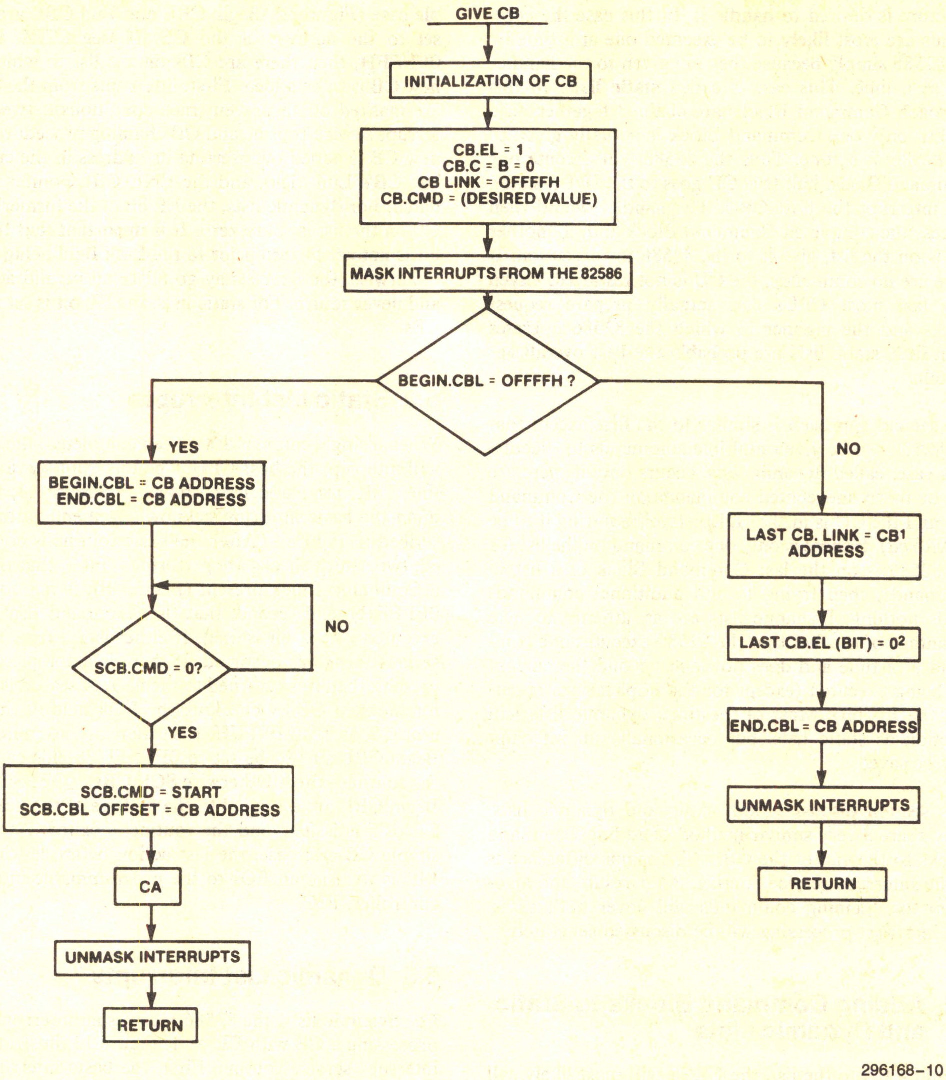
The procedure shown in Figure 10 first clears the CB's C and B bits, sets the I and EL bits to one, and sets the Link field to 0FFFFH. It will then examine Begin.CBL. If Begin.CBL = 0FFFFH, there are no CBs on the list and the process proceeds as in the simple case (Figure 8). Begin.CBL and End.CBL are both set to the address of the CB. If Begin.CBL is not 0FFFFH, then there are CBs on the list to which the new CB is to be added. First, interrupts from the 82586 are masked off to prevent race conditions between interrupt service routine and CB chaining procedure. The new CB is added by inserting its address in the current last CB's Link field, and the END.CBL pointer is updated. For dynamic lists, the EL bit of the formerly last CB on the list is set to zero. **It is important that the EL bit is not set to zero prior to the Link field being valid.** Otherwise, the 82586 may go off to an invalid address and never return. For static lists, the EL bit is set for all CBs.

5.2 Static List Interrupts

When a single command has been completed, the 82586 will interrupt the host CPU. The interrupt service routine, ISR, for static lists, shown in Figure 11, builds upon the basic interrupt acknowledgement routine described in Figure 9. After the basic routine is completed, two sanity checks are performed. First, that there is a Command Block present (Begin.CBL is not equal to 0FFFFH), and second, that the Command Block was executed (the C bit is one). If either is not true, then a serious error has occurred. If there is a completed CB present, then it is returned to the user. A search is made for the next CB by checking the Link field of the current CB. If it is 0FFFFH, then there are no more CBs. Begin.CBL should be set to 0FFFFH in this case and the routine exited. Otherwise SCB.CBL_OFFSET and Begin.CBL are set equal to the address of the next CB, START is issued, and the routine is exited. As in the simple CB add case, the last action before leaving the ISR is to issue an EOI to the programmable interrupt controller, PIC.

5.3 Dynamic List Interrupts

For dynamic lists, the 82586 receives an interrupt after processing a CB with EL = 1. Figure 12 illustrates the interrupt service routine. First, the basic interrupt acknowledgement routine is used from Figure 9. A pointer called CB.pointer is defined to track the CB of immediate interest. CB.pointer is initially set equal to Begin.CBL. It is then tested for the empty condition (0FFFFH), and if empty the routine is exited. If not, the CB itself (pointed to by the CB.pointer) is obtained and the C bit of the CB is examined for completion of execution. If set, the CB.pointer is updated with the location of the next CB (i.e. the CB Link field of the used CB) and the used CB is returned to the handler. The process repeats until the end of the CBL is found. If the C bit was not set, the CU is checked for the IDLE state, and is restarted if required. In either case, EOI is executed, and the procedure is exited.



NOTES:

1. Placement of this statement is critical. For example, if EL is set to zero before CB Link is valid, the 82586 could go to an invalid CB address.
2. For static lists, EL = 1, and this line is not required.

Figure 10. Procedure to Chain CBs: Process One CB at a Time (Static List), or Multiple CBs (Dynamic List)

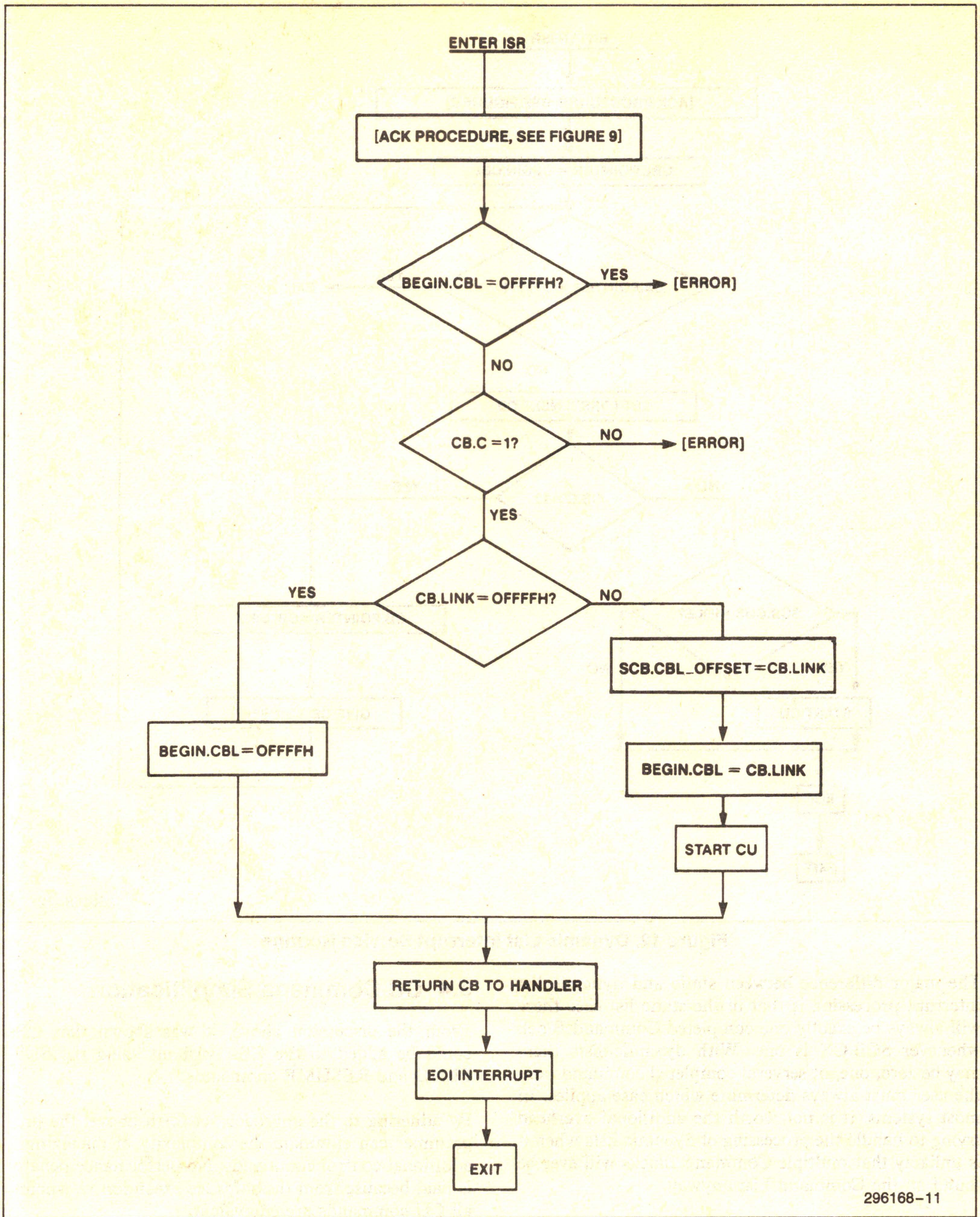


Figure 11. Interrupt Service Procedure for Static Lists (EL Bit Set in all CBs)

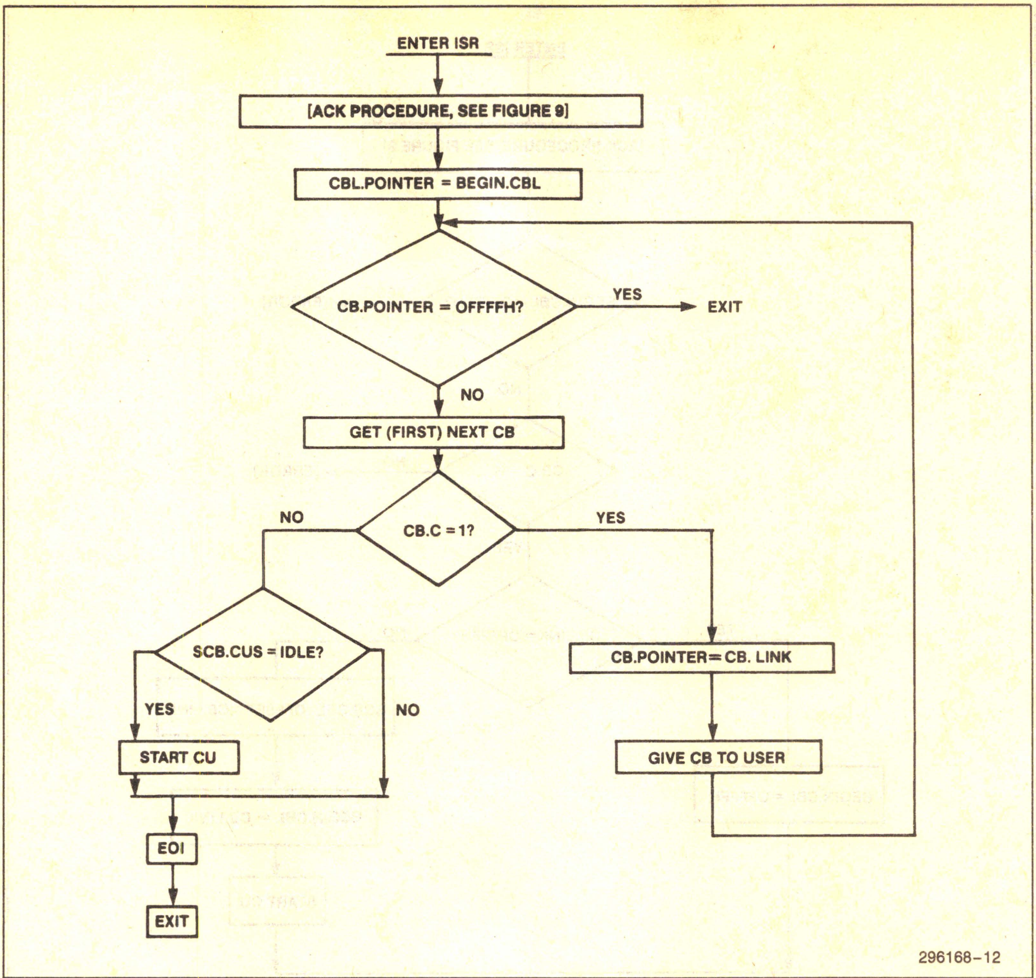


Figure 12. Dynamic List Interrupt Service Routine

The major difference between static and dynamic list interrupt processing is that in the static list case there will always be exactly one completed Command Block whenever SCB.CX is one. With dynamic lists there may be zero, one, or serveral completed commands and the user must always determine which case applies. In most systems, it is not worth the additional overhead trying to handle the processing of dynamic lists when it is unlikely that multiple Command Blocks will ever be found on the Command List anyway.

5.4 CU Command Simplification

From the discussion above, it was shown that CBs could be added to the CBL without using the SUSPEND and RESUME commands.

By adhering to the approach set forth above, the programmer can eliminate the complexity of managing 2 additional control commands. No performance penalty is paid because from the software execution viewpoint, all CU commands are equivalent.

The approach works because when the CU recognizes an EL = 1, it goes directly to the IDLE state without reading the Link field. This rule allows the handler to specify location 0FFFFH as the empty condition, which is helpful in managing CBs.

It will be shown in the next section that RESUME and SUSPEND are not required to append FDs and RBDs to the Receive Data and Free Buffer lists.

6.0 RECEIVE FRAME PROCESSING

The 82586's automatic receive buffer chaining management capability is one of its most powerful features. This capability affords efficient memory usage which in turn can result in fewer lost frames due to lack of buffers. In order to gain the greatest advantage from this capability, the software must be able to dynamically handle interrupts reclaim and add FDs/RBDs to the Receive Frame Area, RFA. 'Dynamically' here means that the CPU need not halt or suspend the RU in order to add either FDs or RBDs.

The RFA can be divided into two lists: the Received Data List, RDL, and the Free Buffer List, FBL (see Figure 13). The RDL is the list of free FDs while the FBL is the list of free RBDs. The basic technique to make additions to the RDL and FBL is similar to adding CBs as described for dynamic lists in section 5.0. As before, use of the RU commands SUSPEND and RESUME is not required.

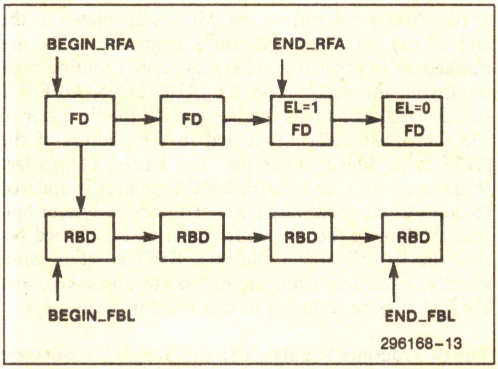


Figure 13. The Receive Frame Area

In order to operate on the RDL and FBL, a pair of pointers is needed for each list. The pointers BEGIN_RFA and END_RFA are used to indicate the beginning and end of the RFA. The pointers BEGIN_FBL and END_FBL are used to indicate the beginning and end of the FBL. **It should be noted that 'beginning' and 'end' here refer to how the CPU views the lists, not the 82586.** As in the case of the chained Command Blocks, the lists that the CPU operates on may contain blocks that the 82586 either has already processed or does not know are present (in the sense of being past the 'last block' as viewed by the 82586). The BEGIN pointers are used by interrupt routines for searching purposes. The END pointers are used to designate a true end of the RDL and FBL.

An interesting problem occurs in receive frame processing that does not occur in CB processing, namely 'orphan Buffer Descriptors.' Consider the case of 20 RBDs, each 128 bytes long, and 3 FDs. If three short frames are received, there will be 17 'RBDs' without any FDs present (they have become 'orphans'). That is, the RBDs do not have a FD to point to them. The following algorithms will also deal with this problem by use of the BEGIN_FBL and END_FBL pointers.

Before presenting the actual algorithms, it is helpful to recall how the RU itself operates on the RBL and FBL. First, the RU never modifies links between individual FDs and between individual RBDs. Second, the RU updates the RBD_OFFSET field in the FD to point to the next free RBD. Third, the RU only reads the RFA_OFFSET field in the SCB at START; the 82586 never modifies this field.

6.1 Supplying FDs to the RDL

The first task of concern is how to place free FDs onto the RDL. Free FDs may be placed on the RDL in two instances. First, at startup time when the 82586 is provided with its initial pool of FDs. Second, when the ULCS is done with a received frame and is returning the FDs and RBDs to the handler. A likely method to give FDs to the handler is to call a subroutine. This routine can be called 'Supply FD.' Supply_FD places new FDs to the end of the RDL, see Figure 14.

Supply_FD first initializes the new FD by setting EL = 1 and LINK_OFFSET and RBD_OFFSET = 0FFFFH. Setting LINK_OFFSET to 0FFFFH indicates that the FD is the last block on the RFA insofar as the CPU is concerned (the 82586 uses EL). **Initializing the RBD OFFSET is critical to avoid system hangups.** If no prefetched RBDs exist when the 82586 is setting up the new FD, the 82586 will read the RBD-OFFSET in the FD expecting to find a pointer to a new RBD. If the RBD-OFFSET = 0FFFFh, then the 82586 assumes that no RBD exists for this FD. If the RBD-OFFSET ≠ 0FFFFh, then the 82586 assumes that a valid RBD exists at the address specified by the RBD-OFFSET.

A check is then made to determine if the RFA is empty (i.e. BEGIN.RFA = 0FFFFH). If it is, the BEGIN and END RFA pointers are set to the address of the FD. The RU is not started because there is only one FD available to the 82586 (see section 6.4). The routine returns back to the handler.

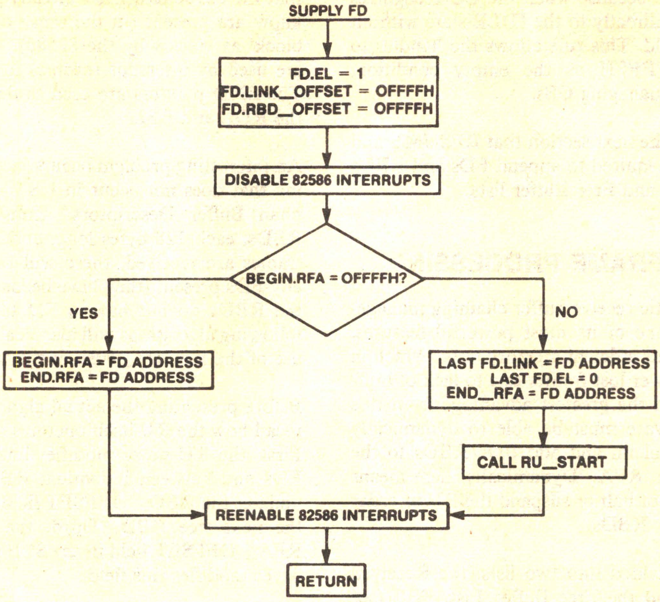


Figure 14. Procedure to Add FDs to the RDL

If the RFA is not empty, the FD is appended to the end of the list. First, interrupts from the 82586 are masked off to prevent an interrupt service routine from modifying the RDL. The last FD.LINK OFFSET (i.e. the FD that was the last one on the list prior to the subroutine call) is updated to the address of the FD to be added to the list. The EL bit of this last FD is set to 0, and the End RFA pointer is updated to point to the 'new' last FD. To prevent system failure, the link to the new FD must be established before the EL bit in the 'old' last FD is set to zero. Finally, interrupts from the 82586 are unmasked, and the RU may be restarted as described in section 6.4.

6.2 Supplying RBDs to the FBL

The process of adding RBDs to the FBL is very similar to that for adding FDs to the RDL. As with the FDs there is likely to be a routine called 'Supply_RBD'. Supply_RBD places new RBDs at the end of the FBL (see Figure 15).

Supply_RBD first initializes the new RBD by setting EL = 1 and LINK OFFSET. Setting LINK_OFFSET to 0FFFFH indicates that the RBD is the last block on the FBL insofar as the CPU is concerned (the 82586 uses EL). A check is then made to determine if the FBL is empty (i.e. Begin FBL = 0FFFFH). If it is, the BEGIN and END FBL pointers are set to the address of the RBD. The RU is not started because there is only one RBD available to the 82586, (see section 6.4). The routine returns back to the handler.

If the FBL is not empty, the RBD is appended to the end of the list. First, interrupts from the 82586 are masked off to prevent an interrupt service routine from modifying the RDL. The last RBD.LINK OFFSET (i.e. the RBD that was the last one on the list prior to the subroutine call) is updated to the address of the RBD to be added to the list. The EL bit of this last RBD is set to 0, and the End RDL pointer is updated to point to the 'new' last RBD. To prevent system failure, the link to the new RBD must be established before the EL bit in the 'old' last RBD is set to zero. Finally, interrupts from the 82586 are unmasked, and the RU may be restarted as described in section 6.4.

The two routines Supply_FD and Supply_RBD were constructed to illustrate the algorithms. In most real systems, it is likely that the actual routines would differ slightly from the ones discussed. The Supply_FD routine would likely accept FDs with a list of attached RBDs. Returning linked FDs and RBDs is useful since most of the time a previously received frame consisting of one RD and at least one RBD will be returned to the handler. Likewise the Supply_RBD routine could also accept a list of RBDs instead of just one. The processing is slightly more complex but takes advantage of the already existing lists instead of first taking the lists apart and then putting them back together again.

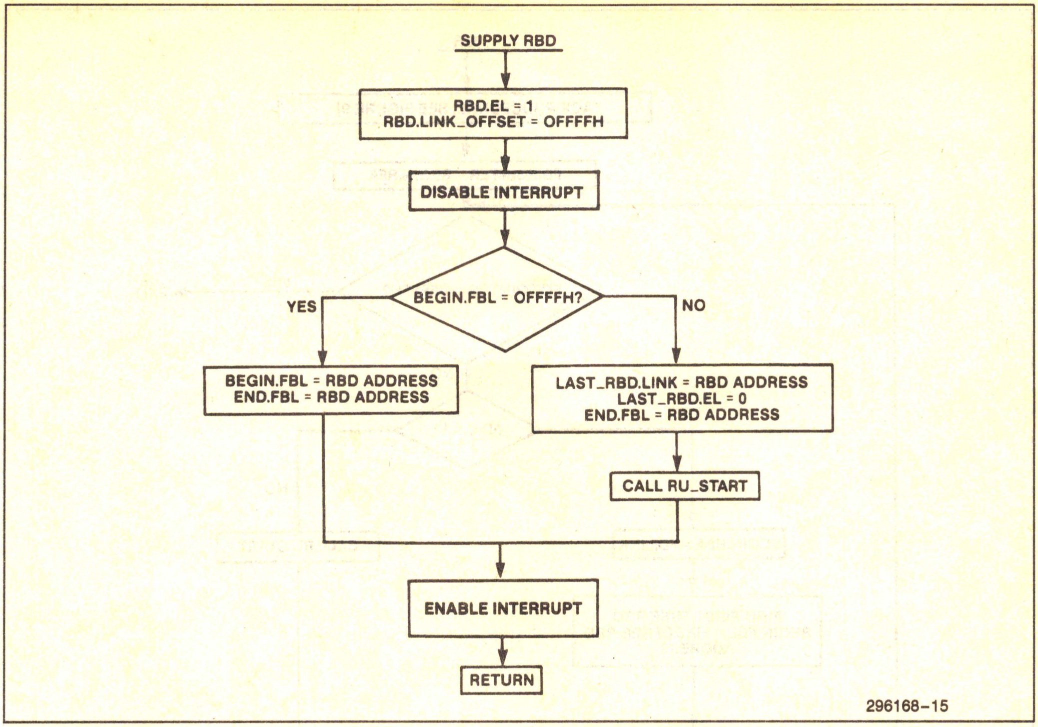


Figure 15. Procedure to Add RBDs to the FBL

6.3 Receive Interrupt Processing

The procedure for handling receive frame interrupts is shown in Figure 16. There are basically three phases to receive interrupt processing. The first phase is to acknowledge receipt of the interrupt from the 82586 (as was done in Figure 9). The second phase is to remove any received frames from the RFA and update the pointers to the RDL and FBL. The third phase is to restart the RU if it is not in the READY state. The first phase is described in section 4.2.

The second phase begins by examining the FR bit in the SCB status field. If it is set, then there may be received frames present. (It should be noted that the word 'may' is used. It is possible that the received frames had been already removed during previous interrupt service processing). To locate received frames the following algorithm is used. A Frame Descriptor pointer, FD.pointer, is defined to access FDs. FD.pointer is initially set to the location of the first known free FD (contained in Begin_RFA). This location is first tested for the empty case (0FFFFH) and if empty, processing is terminated on the FDL. If a FD is present, then it is tested to see if reception of the frame was completed (the C bit is checked). If not, the second phase is completed and processing moves on to the third phase. If the FD's C bit is set, then the location of the next FD is taken from the FD,LINK and stored in Begin_RFA. The new beginning of the FBL is located by examining the RBD pointed at by the RBD_OFFSET field of the completed FD. A search is performed by examining each RBD until the one with EOF set is found. This RBD is the last RBD used by the completed FD. The next RBD (provided that RBD does not also have EL set) is the new first RBD on the FBL. Its address should be stored in Begin_FBL. When this procedure is done, the FD and associated RBDs are forwarded to the user and FD.pointer is set to the next FD. This process continues until either the end of the RDL is found or a FD with the C bit not set is located.

The third phase is concerned with restarting the RU if required. The RU may be in either the IDLE or NO-RESOURCES state and may need to be restarted. The rules for doing this are presented in the next section, 6.4.

6.4 Rules for Starting the RU

Care must be taken when restarting the RU after adding FDs and RDBs. The following rules are helpful to ensure smooth processing:

1) It is unnecessary to restart the RU if it is already running (i.e. in the READY state).

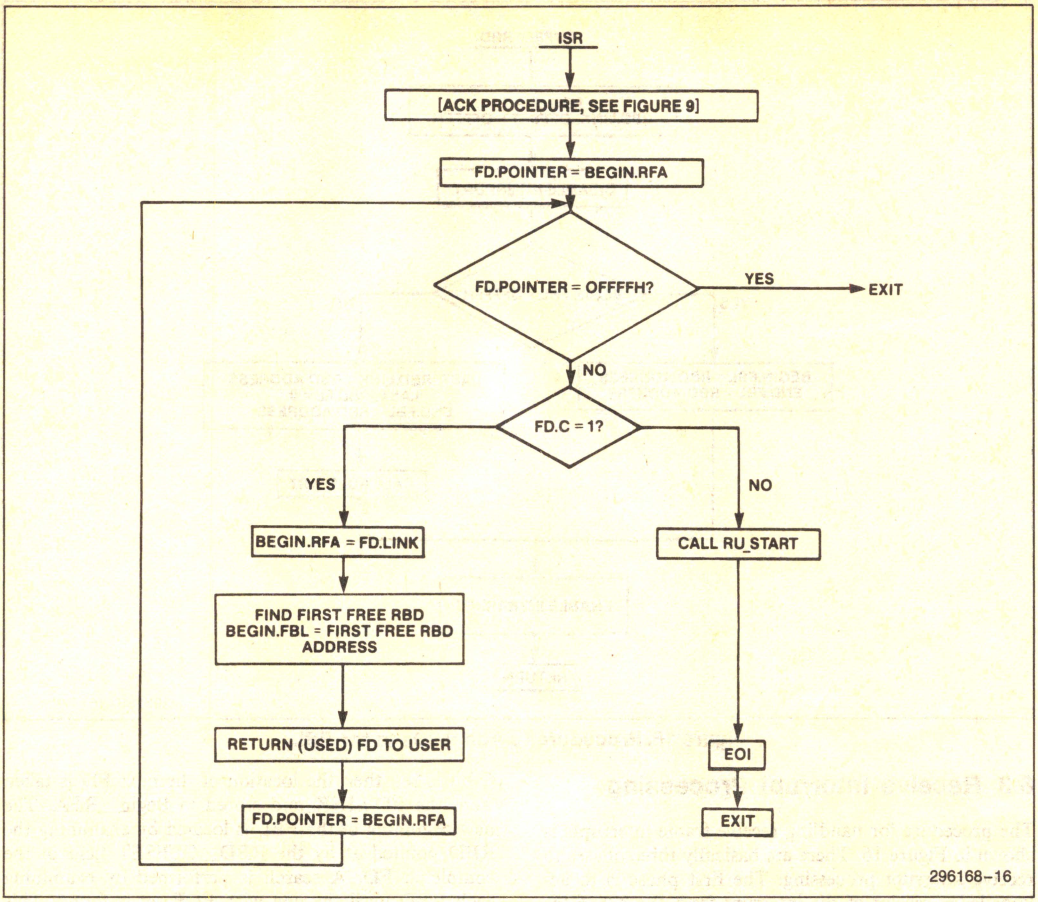


Figure 16. Receive Frame Interrupt Processing Routine

2) There should be at least 2 available FDs on the RDL. For example, if the first FD has EL = 1, then the RU will ignore all remaining FDs. In this case, the RU will go into the NO RESOURCES state after a frame is received. If two or more FDs are available, the RU will stay active because it hasn't seen EL = 1.

3) There should be at least 2 RBDs available to the RU before starting.

A procedure for starting the RU is shown in Figure 17. The procedure begins by checking the rules set forth above (it is assumed that SUSPEND is not used). The RBD_OFFSET of the first FD is set to the beginning of the FBL (using the Begin.FBL pointer). The procedure waits for the SCB command field to be zero; this check ensures that there are no pending commands to the 82586. The RFA offset in the SCB is set to location of the first FD in the list. The RU is then started. Channel Attention is given, and the routine is exited.

6.5 Considerations in Using Receive Buffers

In using the linked buffer structures present with the 82586 there are several important considerations related to how many receive buffers are required and how large should they be. Although the exact numbers will depend upon the performance and available memory in

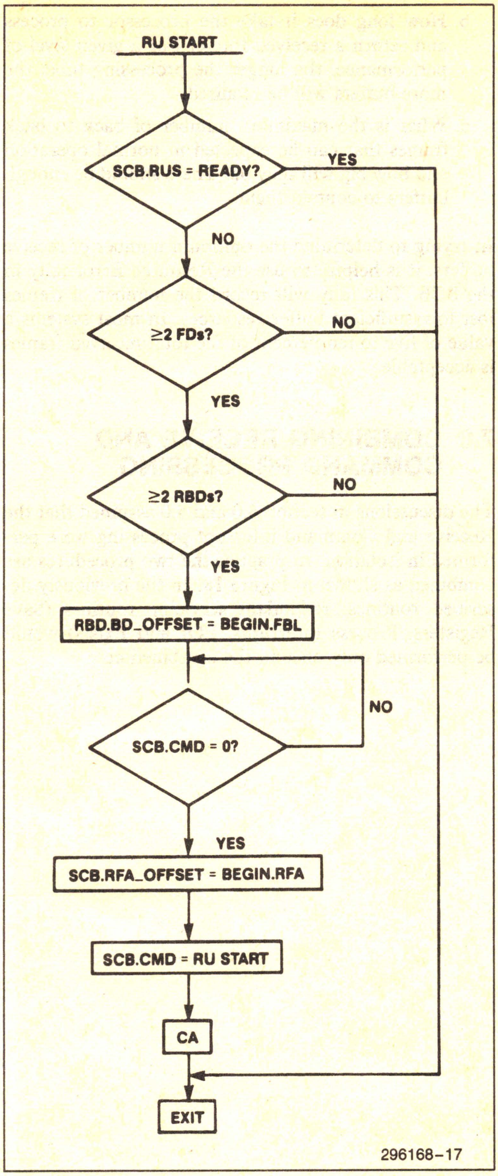


Figure 17. RU Start Procedure

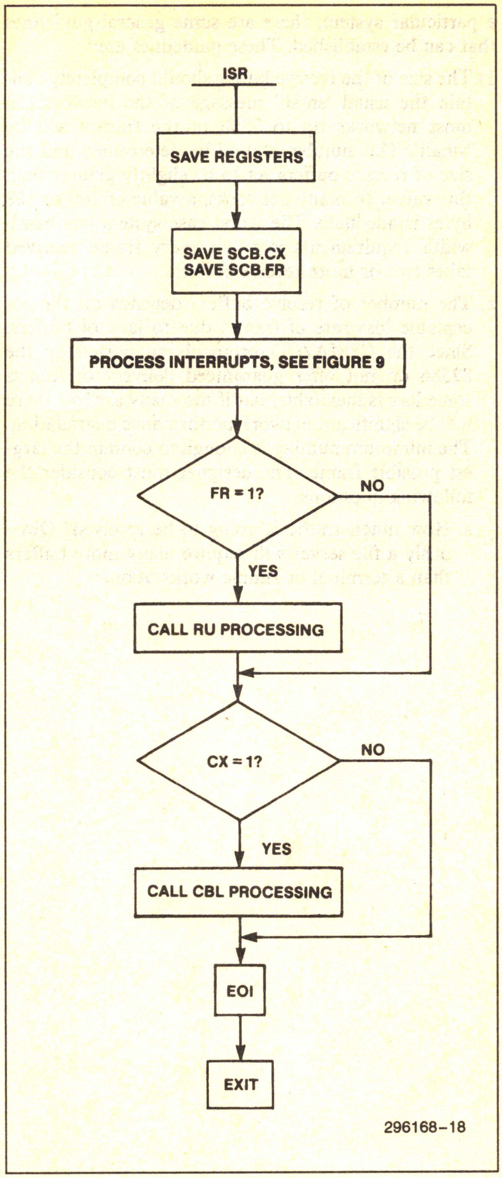


Figure 18. Combined Command and Receive Interrupt Service Routine

a particular system, there are some general guidelines that can be established. These guidelines are:

1) The size of the receive buffer should completely contain the usual 'small' message of the network. In most networks up to 75% of the frames will be 'small'. This number should be determined and the size of receive buffers set to be slightly greater than this value. In many networks, a value of 100 to 128 bytes is adequate. The worst case system bus bandwidth requirements is when every frame received takes two or more receive buffers.

2) The number of receive buffers depends on the acceptable loss rate of frames due to lack of buffers. Since the CSMA/CD protocols supported by the 82586 do not offer guaranteed delivery of frames some loss is inevitable, but if too many are lost there will be significant network performance degradation. The minimum number is enough to contain the largest possible frame. The designer must consider the following questions:

   a. How much traffic is going to be received? Obviously a file server will require many more buffers than a terminal or simple workstation.

   b. How long does it take the processor to process and return a received frame? For a given level of performance, the longer the processing time, the more buffers will be required.

   c. What is the maximum number of back to back frames that can be expected in normal operation and how big will they be? There should be enough buffers to contain them.

In trying to determine the optimum number of receive buffers, it is helpful to use the Resource Error tally in the SCB. This tally will record the number of frames lost to insufficient buffer resources. In most systems a value of five to ten percent of the total received frames is acceptable.

7.0 COMBINING RECEIVE AND COMMAND PROCESSING

The discussions in section 4.0 and 5.0 assumed that the Receive and Command interrupt processing were performed in isolation. In practice the two procedures are combined as shown in Figure 18. In the previously described routines, redundant servicing routines (Save Registers, Process Interrupts, EOI and Exists) would be performed only once in the combined case.

CHAPTER 5
82586 APPLICATIONS

OVERVIEW

This chapter is a collection of brief notes related to system considerations when using the 82586 LAN Coprocessor. The chapter is based on work performed by Intel's Data Communications Application Engineering staff. This work includes computer simulations and actual debugged hardware/software designs.

1.0 MINIMUM 82586 SYSTEM BUS SPEED

82586 Bus bandwidth requirements are an important concern. Parameters that dictate 82586 system bus useage are: buffer size, FIFO-Threshold, interframe spacing, serial data rate, wait states, bus latency, and 8 or 16 bit bus width. The relationship between these parameters is complex.

A worst case analysis can be done to determine the minimum bus frequency that the 82586 must have to receive two or more back to back frames. There are **two** variables of concern. First, the ratio of the parallel bus frequency, Fp, to the serial clock frequency, Fs. The Fp/Fs ratio affects bus latency (i.e. the time it takes for the 82586 to acquire control of the bus), number of wait states (i.e. the additional CPU cycle times required between memory address to valid data received), and handling 82586 buffer/status overhead that can be tolerated before the on-chip FIFOs exceed their capacity.

Second, interframe spacing gives additional time for the 82586 to complete placing received data into memory and handle 82586 buffer/status overhead. Figure 1 shows the timing relationship and activities performed by the serial side and parallel side of the 82586.

Simulations were performed to provide **guidelines** to designers to establish minimum Fs/Fp ratios given various bus latency, wait state, and interframe spacing time conditions. The simulations assumed a system using word mode, and worst case conditions consisting of: last buffer exactly filled up, next buffer remaining empty, and prefetch of next buffer descriptor. The most recently received frame was assumed OK.

The results of these simulations are shown in Figure 2. Note that various wait state (N_W) and bus latency (N_L) environments are represented by sloped lines, according to the empirical formula $16\ N_W + N_L = X$ clock cycles.

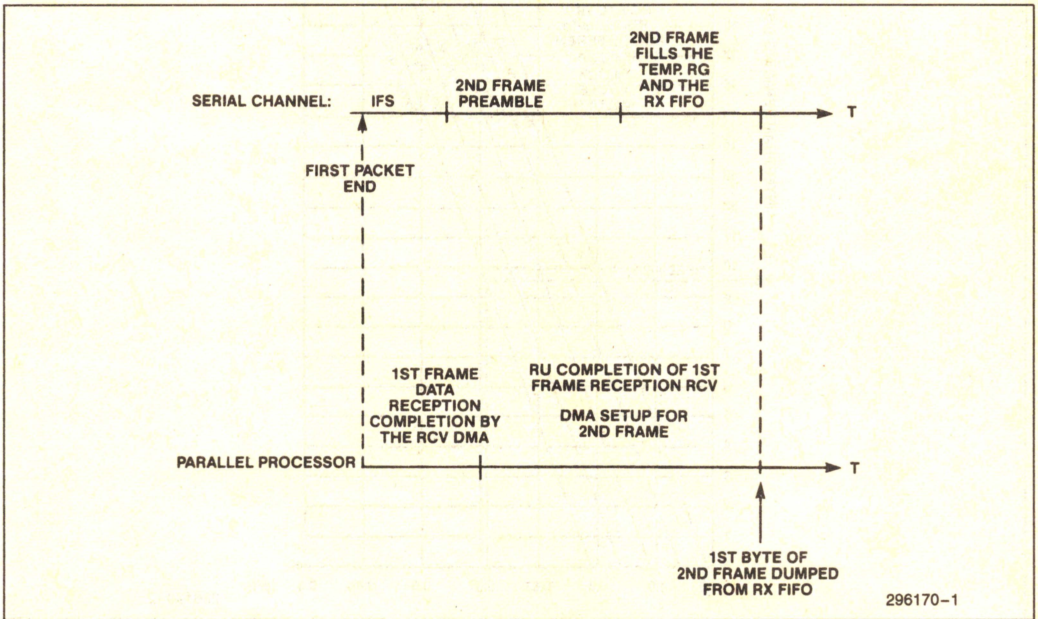


Figure 1. Receive to Receive Interframe Spacing Time Relationships

The following example illustrates use of Figure 2:
Wait states, N_W = 0 clock cycles
Bus latency, N_L = 0 clock cycles
IFS = 9.6 μs

From Table 1, the minimum Fp/Fs is 0.58. In other words, for Fs = 10 MHz, i.e. IEEE 802.3, then Fp can be as slow as 5.8 MHz.

One can verify that for IEEE 802.3, where Fp/Fs = 0.8 (i.e. Fp = 8 MHz), that the 82586 can handle the 9.6 μs interframe spacing (IFS) even when the sum of 16 times the number of wait states (N_W) plus bus latency (N_L) is equal to 80. For example, a 82586 system of 0 wait states, and bus latency of 80 clock cycles can handle back to back frames, separated by the IFS time.

Table 1 summarizes Fp for various bus latency/wait state conditions for IEEE 802.3 from Figure 2.

Table 1. Minimum Bus Frequency for IFS = 9.6μs

N_W (Clock)	N_L (Clock)	Fp (Min) MHz
0	0	5.8
0	5	5.9
1	0	6.2
1	5	6.3
2	0	6.5
2	5	6.8

2.0 SETTING THE 82586 FIFO-THRESHOLD

The 82586 features a programmable FIFO-Threshold. When the threshold value is reached, the 82586 attempts to acquire the system bus via the HOLD/HLDA protocol.

If the FIFO-Threshold is set too low, the 82586 will be constantly accessing the system bus, creating system inefficiencies because the CPU and other system peripherals must perform additional overhead duties related to relinquishing the bus. If the FIFO-Threshold is set too high, the 82586 may not acquire the bus before the FIFO becomes full (or empty), causing received (or transmit) data to be lost. Thus, an optimal threshold setting can be found based on the number of times the 82586 accesses the bus, and ensuring that received data does not get lost.

The variables of concern to setting the FIFO-Threshold are parallel bus frequency (Fp), serial clock frequency (Fs), and the bus latency time (N_L). The ratio of Fp/Fs is important because it quantifies the relationship between how quickly the FIFOs are filled, and how fast the FIFOs can be emptied. The latency time quantifies the amount of time the 82586 must wait before it can begin to empty the FIFOs.

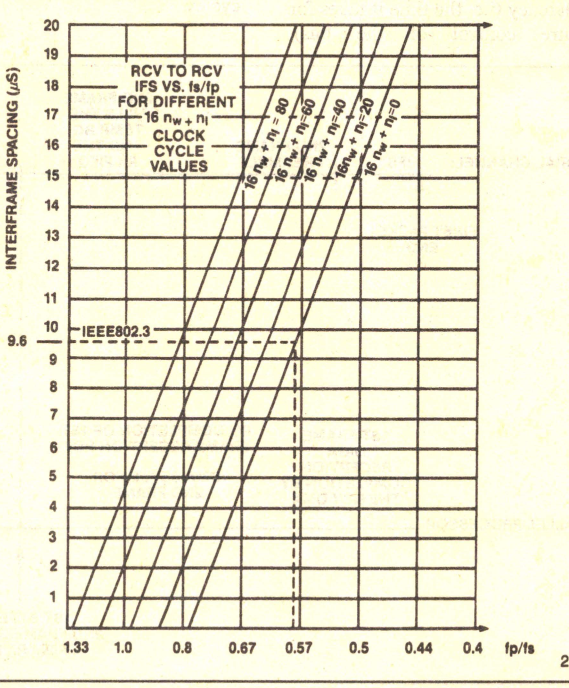


Figure 2. CPU/82586 Parallel Bus Frequency Requirements for given Wait States (N_W) and Bus Latencies (N_L)

82586 system simulations were performed to provide a starting point with which to set an optimal threshold setting. For an actual system, **the FIFO-Threshold setting should be further optimized to accommodate particular application, system and network environments.** The results of the simulations are summarized in Tables 2 and 3. Table 2, the minimum safe FIFO limit presents guidelines for threshold setting, for a given bus latency. Table 3 takes into account time required for the 82586 to perform buffer switching. The results from both tables must be added together to yield the recommended threshold limit.

Consider the following example:
Parallel Bus Frequency, Fp = 8 MHz
Serial Clock Frequency, Fs = 10 MHz
Maximum bus latency, N_L = 4 clock cycles

From Table 2 (Fp/Fs = 0.8, N_L = 4), the minimum safe FIFO limit = 1.

From Table 3 (using N_L = 5), the FIFO limit offset = 4. Taking the sum of the results from Tables 2 and 3, the FIFO-Threshold setting for preliminary evaluation should be 5.

Table 2. Minimum Safe FIFO Limit

Fp/Fs	NL	Minimum FIFO Limit
0.5	0	1
0.5	1	2
0.5	2	2
0.5	3	2
0.5	4	2
0.5	5	3
0.5	8	3
0.5	10	4
0.5	15	5
0.5	20	6
0.5	25	8
0.5	30	9
0.5	40	11
0.8	0	0
0.8	1	0
0.8	2	1
0.8	3	1
0.8	4	1
0.8	5	1
0.8	8	1
0.8	10	2
0.8	15	3
0.8	20	3
0.8	25	4
0.8	30	5
0.8	40	6

NOTES:
Fp = Parallel Bus Frequency
Fs = Serial Clock Frequency
NL = Bus Latency (in cycles)

Table 3. FIFO Limit Offset

Fp/Fs	NL	FIFO Limit Offset
0.4	0	6
0.5	0	5
0.8	0	3
0.4	5	8
0.5	5	6
0.8	5	4
0.4	10	10
0.5	10	8
0.8	10	5
0.4	20	13
0.5	20	10
0.8	20	7
0.4	40	-
0.5	40	15
0.8	40	10

3.0 THE MINIMUM BUFFER SIZE

The 82586 employs memory buffer chaining to ensure efficient use of memory.

From a network point of view, the minimum buffer size should be no less than the largest 'short' (control) frame plus a few bytes. This practice will eliminate unnecessary buffer linking. A minimum of 64 to 128 bytes is usually sufficient.

From the 82586 point of view, the upper bound of receive buffer size is a function of the network environment and system memory constraints; **the lower bound is a function of the 82586's ability to prefetch receive buffer descriptors and buffer fill time.** In particular, while the 82586 is filling up a buffer, the 82586 will prefetch the next receive buffer descriptor. Thus, the prefetch must be completed before the current buffer becomes full.

The variables affected receive buffer fill time, and consequently minimum buffer size, are the parallel bus frequency, Fp, and serial clock frequency, Fs, bus width, and wait states, N_W. Computer simulations were performed on a general system to provide designers with the minimum receive buffer size. The results are shown in Table 4.

The following example illustrates use of Table 4.

Given:

- Parallel bus frequency, Fp = 8 MHz
- Serial clock frequency, Fs, = 10 MHz
- Wait states, N_W = 1
- Bus width = 16 bits (word mode)

From Table 4, the minimum receive buffer size is 20 bytes.

Note that from Table 4 that the largest minimum is 55 bytes. Thus, it can be concluded that in general receive buffers in excess of 64 bytes are sufficient for the 82586 to handle linked lists in hardware.

Table 4. Minimum Buffer Size

	$N_W = 0$		$N_W = 1$		$N_W = 2$	
Fp/Fs	Word Mode	Byte Mode	Word Mode	Byte Mode	Word Mode	Byte Mode
0.4	51	*	55	*	42	*
0.5	47	26	51	22	42	20
0.8	36	24	34	22	30	20
1	36	24	34	22	30	20

NOTES:
*Will always underrun or overrun
Fp = Parallel Bus Frequency
Fs = Serial Clock Frequency

The minimum transmit buffer size is determined by the same factors as receive buffers plus end-of-frame-processing (updating status, pointers, BD count, and statistics; setting up the RU for the next reception; etc.). The 82586 can underrun if the values of Table 4 are used for the first transmit buffer of a frame to be transmitted. Underruns will occur when the 82586 is issued a Transmit command just prior to or during a receive operation. In this case, during reception the 82586 will fetch the first TBD and fill the Transmit FIFO. After the completed reception, the 82586 will start transmission and simultaneously complete end-of-frame-processing. End-of-frame-processing will delay the prefetch of the second TBD, and cause the underrun. Thus the minimum length of the first Transmit Buffer must be long enough so that the next TBD prefetch will occur before the first buffer is transmitted. For systems with Fp/Fs = 0.6 a first transmit buffer in excess of 75 bytes is usually sufficient to avoid underruns; for systems with Fp/Fs = 0.8, the first transmit buffer should be in excess of 54 bytes. The succeeding transmit buffers for a single frame can follow the recommendations of Table 4.

The 82586 supports simultaneous transmit and end-of-frame-processing operations to ensure that the 82586 will be able to transmit soon after a receive operation. Thereby avoiding the case where a receiving station continually gets beaten onto the network by other stations. This case is important in communication intensive applications like file servers.

4.0 SYSTEM CONFIGURATIONS

4.1 80186 Elementary Maximum Mode System

SYSTEM INTERFACE

The 82586 does not require any 'TTL glue' to interface to an 80186 microprocessor bus. Thus, it is highly recommended for minimum component count communication systems (see Figure 3). The 82586 is configured to Maximum Mode by strapping the MN/$\overline{MX}$ pin to ground; in this mode the 82586 generates status signals that will be used for bus control signal generation.

The 82586 is clocked by the CLKOUT signal generated by 80186. Thus, the existing address latches, data transceivers and bus controller can be shared by the CPU and the 82586.

The $\overline{S0}$, $\overline{S1}$ signals of the 80186 and 82586 are wired together, driving the 8288 bus controller. The 80186 and 82586 have internal pullups so no external resistors are required on the $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ lines. The 8288's $\overline{S2}$ input is driven only from the 80186; this status is not generated by the 82586 because it accesses memory only (no I/O). The $\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ lines are used by the 8288 to generate a full set of standard bus control signals.

The shared data bus of the 80186 and the 82586 is 16 bits wide. The system memory can be accessed either by the 80186 or by the 82586. Bus arbitration is resolved by the HOLD/HLDA protocol. The CPU grants the bus to the 82586 by issuing the HLDA high as a response to bus request from the 82586 (HOLD high). The CPU is able to withdraw its bus grant by dropping HLDA low. In this case the 82586 will release the bus within a maximum four clock cycles in word mode.

Care must be taken so that the CPU does not take away HLDA during transmission or reception because there will be a high chance of frame abortion by underrun or overrun respectively.

Communication on the task level between the CPU and the 82586 is accomplished by a shared system memory mailbox, the System Control Block, and the CA/INT handshake.

It is possible to create a multimaster system by using the 8289 bus arbiter. It is suggested that in these systems the 82586 communication node is assigned the highest priority. A high priority will ensure a minimum bus latency for the 82586 whenever it needs the bus, thus avoiding waste of bus bandwidth by underrun or overrun frames.

To realize the IEEE 802.3 standard (i.e. 10 Mbps serial data rate and 9.6 μs Interframe Spacing), it is sufficient to operate at 8 MHz system clock and zero wait state bus cycles. Also, lower rates and slower memories (that introduce wait state) are possible in IEEE 802.3 systems, see section 1.

SERIAL INTERFACE

Figure 3 displays an 80186 based Elementary Maximum Mode System, containing an Intel 82501 Ethernet Serial Interface.

4.2 Stand Alone Multibus System

SYSTEM INTERFACE

It is possible for an 82586 CPU system to share a memory interface via the Multibus as shown in Figure 4. The Multibus is the Intel's standard bus structure which allows Intel's board products to communicate. Standard address latches, data transceivers and a bus controller are needed in order to interface to a demultiplexed Multibus.

The 8289 Bus Arbiter is needed to resolve Multibus arbitration. It is recommended to assign the 82586 system the highest priority in order to efficiently handle communication tasks. The highest priority interrupt request on the Multibus, INT0, is used to inform the remote CPU that a communications task was completed.

4.3 Dual Port RAM Systems

SYSTEM INTERFACE

The 82586 operating at 10 Mbps serial bit rate, may utilize a significant percentage of bus bandwidth, thus leaving insufficient bus bandwidth for other purposes. When the 82586 bus bandwidth utilization is intolerable, a dual port memory system with one port dedicated to the 82586 is recommended. Figure 7 displays a typical dual port memory based system.

The basic building blocks for the dual port RAM system are standard dynamic RAM's and the 8207 DRAM Controller. The 8207 provides not only dynamic RAM refresh, but also arbitrates between each of the process requests and directs data to or from the appropriate port. The 8207 has a LOCK capability that enables the 82586 to lock out CPU accesses while the 82586 finishes descriptor and error tally processing.

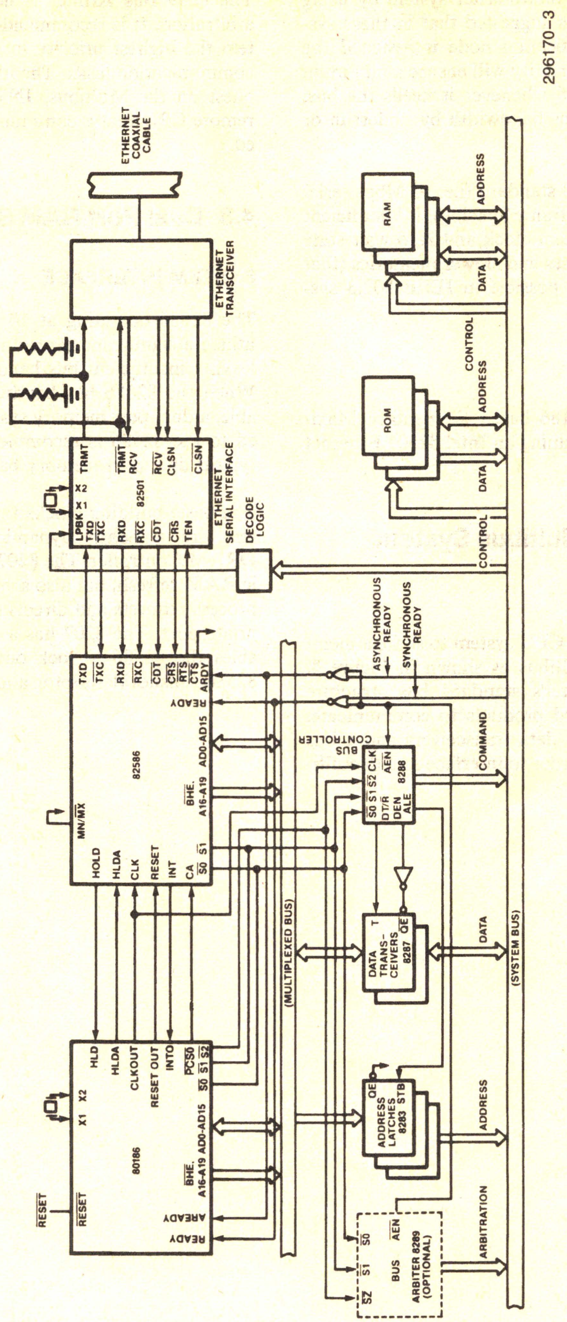


Figure 3. 80186 Elementary Maximum Mode System

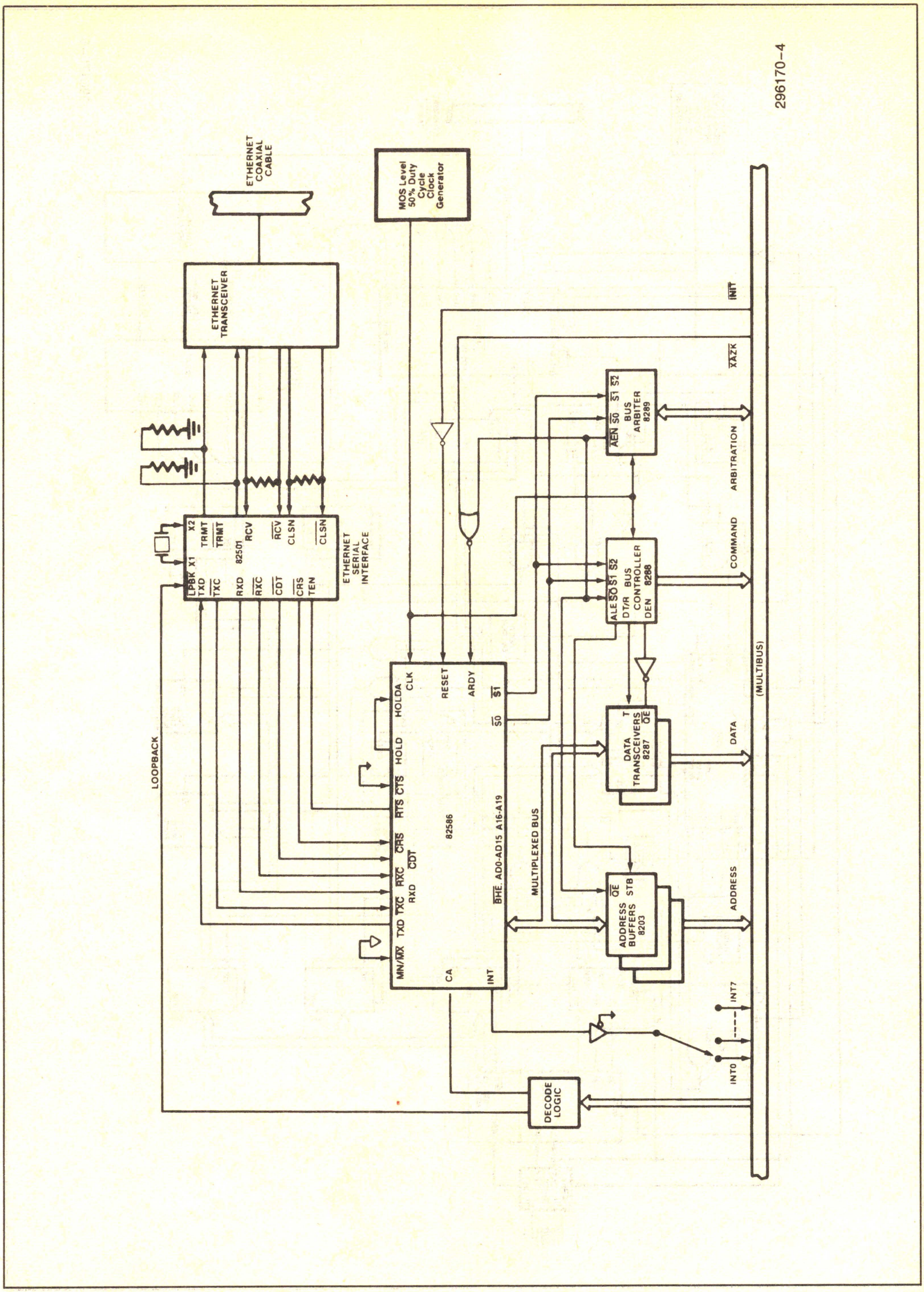


Figure 4. 82586 Stand Alone Multibus Interface

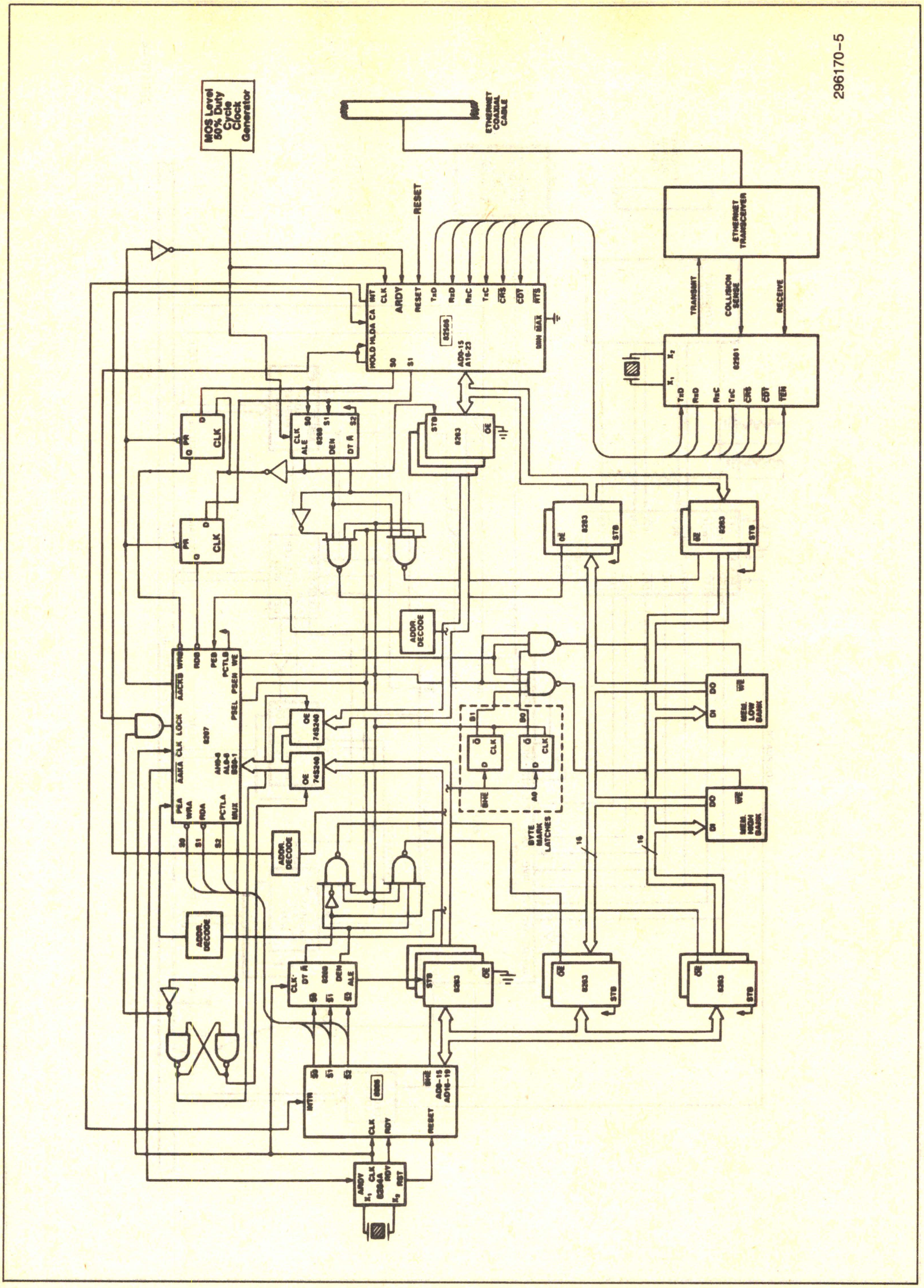


Figure 5. 82586 Used in a Dual Port Configuration

The dual port RAM consists of two memory banks (LOW and HIGH), eight 8283 latches, their steering logic (four NAND gates plus two inverters), 8207, a MUX, and a memory read/write control logic (BYTE MARK LATCHes plus two NAND gates). The port is selected to read/write from/to the memory via PSEL, PSEN 8207 generated signals and DEN signals generated by both ports systems. The LOW or HIGH bank is selected by the BYTE MARK LATCH (driven by A0, $\overline{\text{BHE}}$ and PSEN). The transfer direction (read or write) is selected via the WE signal from the 8207 and DT/$\overline{\text{R}}$ signals generated by both port systems. The port arbitration is provided by the 8207, that is strobing-in both port addresses via a MUX and outputs the presently selected port address.

The strobed-in address is chosen by the 8207 generated MUX signal. Once the processor commanded transfer is executed, an $\overline{\text{XACK}}$ signal is issued by the 8207 to acknowledge transfer completion. As shown in Figure 5 configuration, the 82586 has absolute control of the dual ported memory, immediately after issuing the HOLD signal. This is accomplished by the 8207 LOCK signal driven by the 82586 HOLD.

The 82586 port system shown in Figure 5 is similar to the MULTIBUS stand-alone configuration. Thus a dedicated MOS level Clock Generator is needed on the 82586 port.

NOTE:
If a 22 bit address space is sufficient, the 82586 can be configured to Minimum Mode (strapping the MN/$\overline{\text{MX}}$ pin to VCC). In this mode, the 82586 generates directly the DEN and DT/$\overline{\text{R}}$ signals, thus saving the 8288 Bus Controller.

A low cost dual port RAM design is discussed in section 6.

4.4 Multiple Bus Master Systems

The 82586 is commonly used in systems with other bus masters, for example, Intel's Text Coprocessor, the 82730, or multiple 82586s. To ensure that these bus masters are initialized separately, the following procedure is recommended.

1) Reset the system through a hardware RESET.
2) Set up the SCP, ISCP for Master Number 1, with appropriate pointers.
3) Give Channel Attention to Master Number 1.
4) Change the ISCP SCB OFFSET address in ISCP for Master Number 2.
5) Give Channel Attention to Master Number 2.

Steps 4 and 5 may be repeated for the third or more master on the same bus.

5.0 CALCULATING UNIQUE MULTICAST ADDRESSES

The 82586 performs multicast address filtering through a hashing algorithm. The CRC polynomial is used to map the received multicast address into a hash table consisting of 64 bits. Because the CRC polynomial is used, it is possible for two or more multicast addresses to be mapped into the same bit in the hash table.

It is possible to select up to 64 unique multicast addresses. The Pascal program Hash Calculation, shown in Table 5, selects unique addresses. The user may specify either the 16 or 32-bit CRC function, and variable address length (1 to 6 bytes). A warning will be issued for non-multicast addresses (i.e. starting with a 0 bit). The function HASH computes the Hash value which is the bit address in the 82586 Hash table.

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT)

```
TYPE MCtype                        = ARRAY [0..47] OF BOOLEAN; (* bit pattern
                                             that represents the Multicast Address.
                                             0 is the least significant bit.
                                             Addresses shorter than 6 bytes are
                                             justified to the least significant bit. *)
     HTtype                        = 0..63;         (* entry of hash table *)

VAR AddressLength                  : 1..6;         (* Address Length *)
     CRCLength                     : INTEGER;      (* if 16 then CRC 16 else CRC 32*)
     MCV                           : MCtype;       (* Multicast Address Vector *)
     k, l, m                          : INTEGER;      (* temp *)
     digit                         : CHAR;         (* hexadecimal digit *)
     numb                          : INTEGER;      (* numerical equivalence *)

FUNCTION HASH (AL                                 (* Address Length in bytes *)
               CL:INTEGER;                        (* CRC length: 16 or 32 bits *)
               X:MCtype)                          (* bit array that represents Multicast
                                                     Address *)
               : HTtype;                          (* result: HASH value between 0 and 63 *)
VAR
     A         : ARRAY [0..31] OF BOOLEAN; (* the CRC generator *)
     FB        : BOOLEAN;                 (* Feedback *)
     i         : INTEGER;                 (* temp *)
     Y         : HTtype;                  (* Y = Hash(X) *)

     BEGIN      (* this function returns the hashed value of X.
                   according to the Address Length and the CRC Length *)

     FOR i:=0 TO 31 DO A[i] := true; (* initialize to all ones *)

     FOR i:=0 TO AL*8-1 DO  (* repeat according to the address length *)
          BEGIN
          IF CL=16 THEN   (* separate CRC computation for CRC-16 *)
               BEGIN
               FB := A[15] <> X[i];          (* compute feedback *)
                                             (* shift the CRC register *)
               A[15] := A[14];
               A[14] := A[13];
               A[13] := A[12];
               A[12] := A[11] <> FB;         (* exclusive or *)
               A[11] := A[10];
               A[10] := A[9];
               A[9]  := A[8];
               A[8]  := A[7];
               A[7]  := A[6];
               A[6]  := A[5];
               A[5]  := A[4] <> FB;
               A[4]  := A[3];
               A[3]  := A[2];
               A[2]  := A[1];
               A[1]  := A[0];
               A[0]  := FB;
               END
```

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT) (Continued)

```
ELSE
     BEGIN        (* separate computation for CRC-32 *)
     FB := A[31] <> X[i];       (* compute feedback *)
                                (* shift the CRC register *)
     A[31] := A[30];
     A[30] := A[29];
     A[29] := A[28];
     A[28] := A[27];
     A[27] := A[26];
     A[26] := A[25] <> FB;
     A[25] := A[24];
     A[24] := A[23];
     A[23] := A[22] <> FB;
     A[22] := A[21] <> FB;
     A[21] := A[20];
     A[20] := A[19];
     A[19] := A[18];
     A[18] := A[17];
     A[17] := A[16];
     A[16] := A[15] <> FB;
     A[15] := A[14];
     A[14] := A[13];
     A[13] := A[12];
     A[12] := A[11] <> FB;
     A[11] := A[10] <> FB;
     A[10] := A[9] <> FB;
     A[9]  := A[8];
     A[8]  := A[7] <> FB;
     A[7]  := A[6] <> FB;
     A[6]  := A[5];
     A[5]  := A[4] <> FB;
     A[4]  := A[3] <> FB;
     A[3]  := A[2];
     A[2]  := A[1] <> FB;
     A[1]  := A[0] <> FB;
     A[0]  := FB;
     END;

END;        (* of CRC calculation *)

            (* select the 6 bits out of the CRC register *)
IF A[5] THEN Y := 1 ELSE Y := 0;
IF A[6] THEN Y := Y + 2;
IF A[7] THEN Y := Y + 4;
IF A[2] THEN Y := Y + 8;
IF A[3] THEN Y := Y + 16;
IF A[4] THEN Y := Y + 32;
HASH := Y;   (* return the value *)
END; (*of hash function *)
```

Table 5. PROGRAM Hash Calculation (INPUT, OUTPUT) (Continued)

```
BEGIN       (* main program that interacts with the user *)
WRITELN ('This program calculates the hash function of Multicast addresses');
WRITELN ('Type CTRL/Y for exiting the program')'
WRITELN(' ');
WRITE('Enter CRC length (16 for CRC-16) >> ');
READLN(CRCLength);
WRITE ('Enter Address length (in bytes) >> ');
READLN(AddressLength);
IF AddressLength>6 THEN WRITELN(' Out of range !')

ELSE BEGIN

REPEAT      (* for each Multicast Address in the set *)

FOR k:=0 TO 47 DO MCV[k] := FALSE;      (* initialize the multicast vector *)

WRITELN('Enter Multicast Address (exactly ', AddressLength*2:3,' hexadecimal
  digits');
WRITE('starting with MOST significant) >> ');

FOR k:= AddressLength*2-1 DOWN TO 0 DO          (* for all digits *)
     BEGIN                                      (* read a hexadecimal digit *)
     READ(digit);                               (* and convert it to binary *)
     IF (48<=ORD(digit)) AND (ORD(digit)<58) THEN numb := (ORD(digit)-48)
ELSE
     IF (65<=ORD(digit)) AND (ORD(digit)<71) THEN numb := (ORD(digit)-55)
ELSE
     IF (97<=ORD(digit)) AND (ORD(digit)<103) THEN numb := (ORD(digit)-87);

     FOR l:=0 TO 3 DO
           BEGIN                                (* convert the digit to binary and
                                                insert it to the vector *)
           IF ODD(numb) THEN MCV[4*k+l] := TRUE ELSE MCV[4*k+l] := FALSE;
           numb := numb DIV 2;
           END;
     END;
     READLN;                                    (* flush the line *)
     IF NOT MCV[0] THEN WRITELN ('not a multicast address!');

                                                (* call the hash function *)
     WRITELN('HASH = ',HASH(AddressLength, CRCLength, MCV));

UNTIL FALSE;    (* for ever *)
END;            (* of loop for specific Multicast Address *)
END.
```

6.0 A LOW COST DUAL PORT MEMORY DESIGN

The 82586 is a bus master designed to share the CPU system bus. The shared bus configuration affords systems that use the fewest components and lower cost. However, in some applications the bus bandwidth required by the 82586 may seriously degrade system performance. The solution to this problem is a dual port memory between the host CPU and the 82586. Figure 6 illustrates a shared bus system and Figure 7 shows a dual port 82586/CPU system configuration.

In 82586 applications the dual port approach is useful in the following cases:

1) To minimize bus contention between the CPU and the 82586. The 82586 bus interface design is free of bus latency concerns. This issue is important in systems where the peripherals attached to the system bus cause latency that cannot be tolerated by the 82586, or vice versa.
2) To interface to a limited bandwidth bus. The 82586 may consume as much as 2M bytes/s bandwidth while transmitting or receiving data at 10 Mbps. The instantaneous bandwidth required to meet the IEEE 802.3 specification is even greater. In a dual port system the host CPU is able to access memory when the 82586 is accessing memory, thus high system performance is maintained. This issue is particularly important in 8-bit bus systems.
3) To simplify the interface between the 82586 and non-Intel CPUs.

This section describes an example of dual port design using the 82586 with an 80186.

6.1 Hardware Design

The major characteristics of the hardware design are as follows:

Dual Port Memory Size	-	8K bytes
Memory Type	-	Static RAM's
CPU	-	80186
Bus Arbitration Logic	-	TTL SSI/MSI chips
82586 Bus Speed	-	8 MHz
82586 Serial Data Rate	-	10 Mbps

The objective of this hardware design is to keep the design simple and inexpensive. Figure 8 shows a block diagram of the hardware. The CPU interface is general enough to simplify the interface to CPU's with multiplexed or demultiplexed data/address buses. The size of the system RAM is a function of the following factors:

1) Network traffic size.

2) Net traffic pattern (size of frames).

3) Other CPU tasks.

The memory size is 8K bytes to satisfy typical needs. However, the hardware accommodates expansion of the dual port RAM. Static RAMs are used to keep the design simple and low cost.

The 82586 is set to Minimum Mode to keep the chip count low. The CPU for the example is the 80186. Figure 9 shows a general purpose CPU interface.

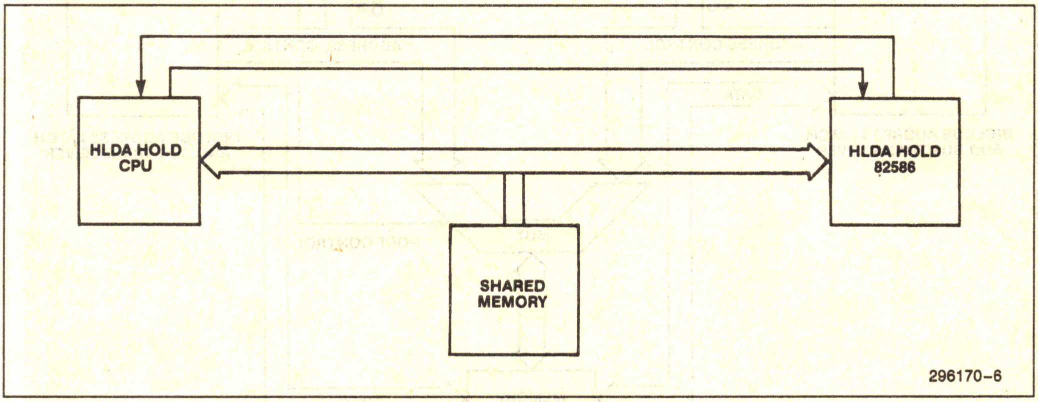


Figure 6. A Typical Shared Memory Design Using the 82586

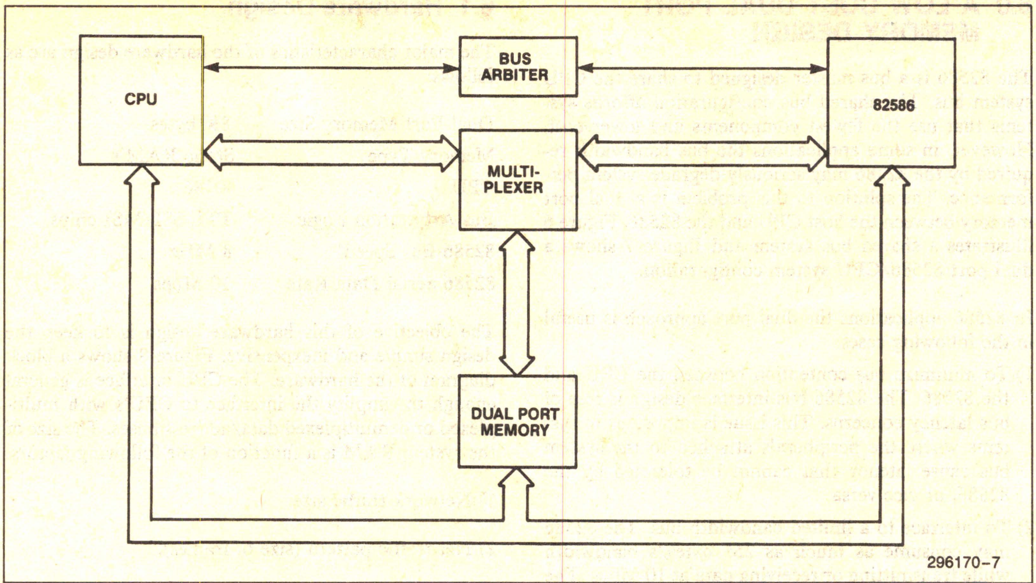


Figure 7. A Typical Dual Port Memory Design Using the 82586

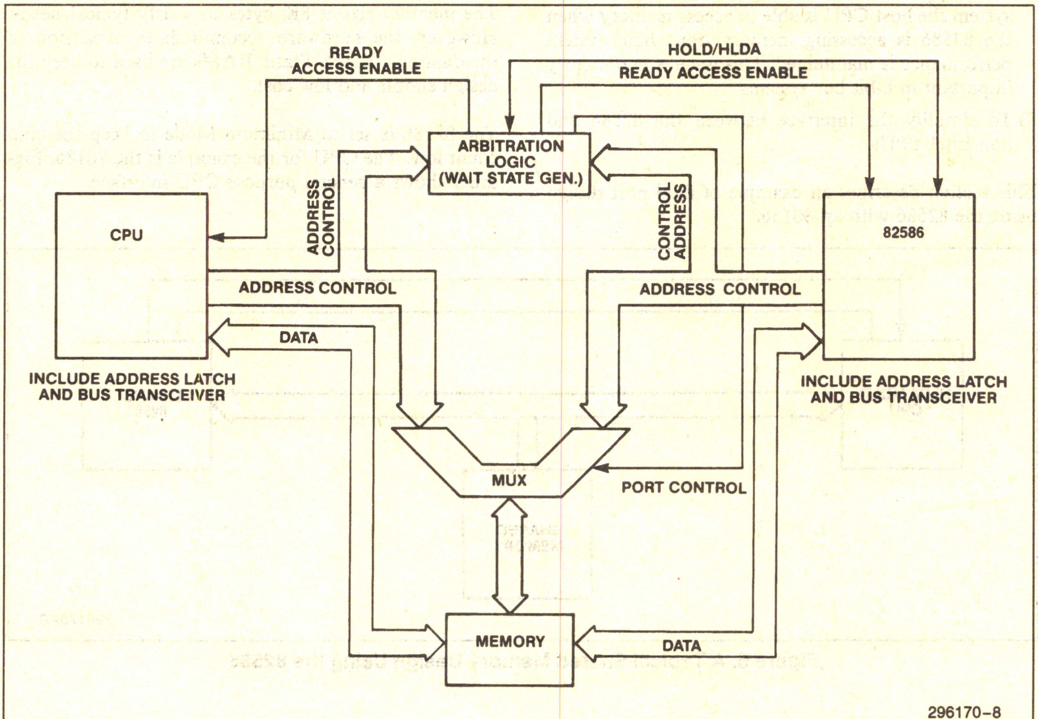


Figure 8. Hardware Block Diagram

BUS ARBITRATION LOGIC

Bus contention between the 82586 and the 80186 is resolved on the basis of the following principles:

1) When the 82586 is accessing the Dual Port Memory RAM, the CPU is prevented from accessing the memory by inhibiting the Ready signal.

2) The CPU can access memory whenever the 82586 is not accessing it.

3) The 82586 gains control of the bus within four to five clocks after a request, and the CPU is put into the wait mode.

The sequence of events when the 82586 requests the bus is:

1) The 82586 requests the bus by activating the HOLD signal.

2) The HLDA is given to the 82586 after five clocks max.

3) The Ready signal to the CPU is disabled within one clock of the 82586 HOLD request.

4) The CPU is isolated from the dual port RAM by disabling the data buffers within 2 clocks of the 82586 HOLD request.

5) The multiplexer is switched to the 82586 bus.

6) If the CPU is accessing the dual port RAM at the time of the 82586 bus request, the CPU is either placed in a wait mode, or it is given enough time to complete its bus cycle.

7) If the CPU has been placed in a wait mode, it remains there until the 82586 is done with the Dual Port RAM. Then the CPU's data and address buffers are enabled, the Ready signal is enabled and the multiplexers are switched back to the CPU, thus allowing the CPU enough time to complete its cycle.

The Channel Attention for the 82586 is generated using a peripheral chip select line of the 80186. In this application, the interrupts from the 82586 are ignored. The diagnostic software, discussed in section 6.2, works under an interactive polled environment.

READY GENERATION

The design adopts a simple scheme to generate the Ready signal for the 82586 and the 80186. The Ready is generated when either the Read or Write signal goes active. The 82586 is given the Ready signal if it is holding the bus (HOLD active) and either Read or Write signal is active. During this period the Ready signal going to the 80186 Ready generation logic is disabled.

The Dual Port RAM generates its own Ready signal for the 80186. This Ready signal is only generated when the 80186 accesses the Dual Port RAM and the 82586 HOLD is inactive. Other memory blocks and peripherals should generate their own Ready signal. Thus all the Ready signals going to the 80186 can be ORed together provided that they are normally low.

Asynchronous Ready is used for both 82586 and 80186 for ease of implementation. This design requires zero wait states for both 82586 and 80186 access. However, for slower memories, a wait state generator may be added to the Ready generation logic.

SETTING THE FIFO LIMIT

When the 82586 needs the bus, it activates its HOLD signal and waits for the HOLD Acknowledge signal (HLDA) before starting any memory access cycles. In the case when the CPU and the 82586 are on the same bus, the HOLD signal from 82586 will go to the HOLD input of the CPU. The CPU will eventually grant the HOLD Acknowledge (HLDA) to the 82586. The time to grant HLDA is dependent on the CPU and the task it is executing at the time of the HOLD request. When HLDA is granted, the 82586 starts its memory access operations.

The 82586 Receive and Transmit FIFO-Threshold limits are set as a function of the HOLD-HLDA delay. The following Receive case illustrates how the FIFO-Thresholds are set. Assume that the Receive FIFO trigger is set at 6. This setting will ensure that the 82586 will make a bus request when 6 bytes have been placed into the Receive FIFO. Since the FIFO is 16 bytes deep, the 82586 must start emptying out the FIFO before the FIFO fills up. In this example, the 82586 must acquire the bus within 10 bytes time (16 − 6 bytes) to avoid an overrun condition. At 10 Mbps, 10 bytes equates to 8 microseconds or 64 CPU clocks (assuming the CPU is running at 8 MHz clock rate). Depending upon the application environment, granting the bus to the 82586 in 64 clock times may or may not be easy to accomplish.

The dual port design avoids the problems associated with bus latency. The HOLD Acknowledge is given to the 82586 within 5 clock cycles of the HOLD request, thus allowing for very small HOLD-HLDA delays.

The 82586 operates in a burst mode when it is reading or writing to the memory. The length of the burst is a function of the number of bytes in the Receive or Transmit FIFO at the time the 82586 gets the HOLD Acknowledge signal. When the 82586 obtains control of the bus, it empties the entire contents of the FIFOs.

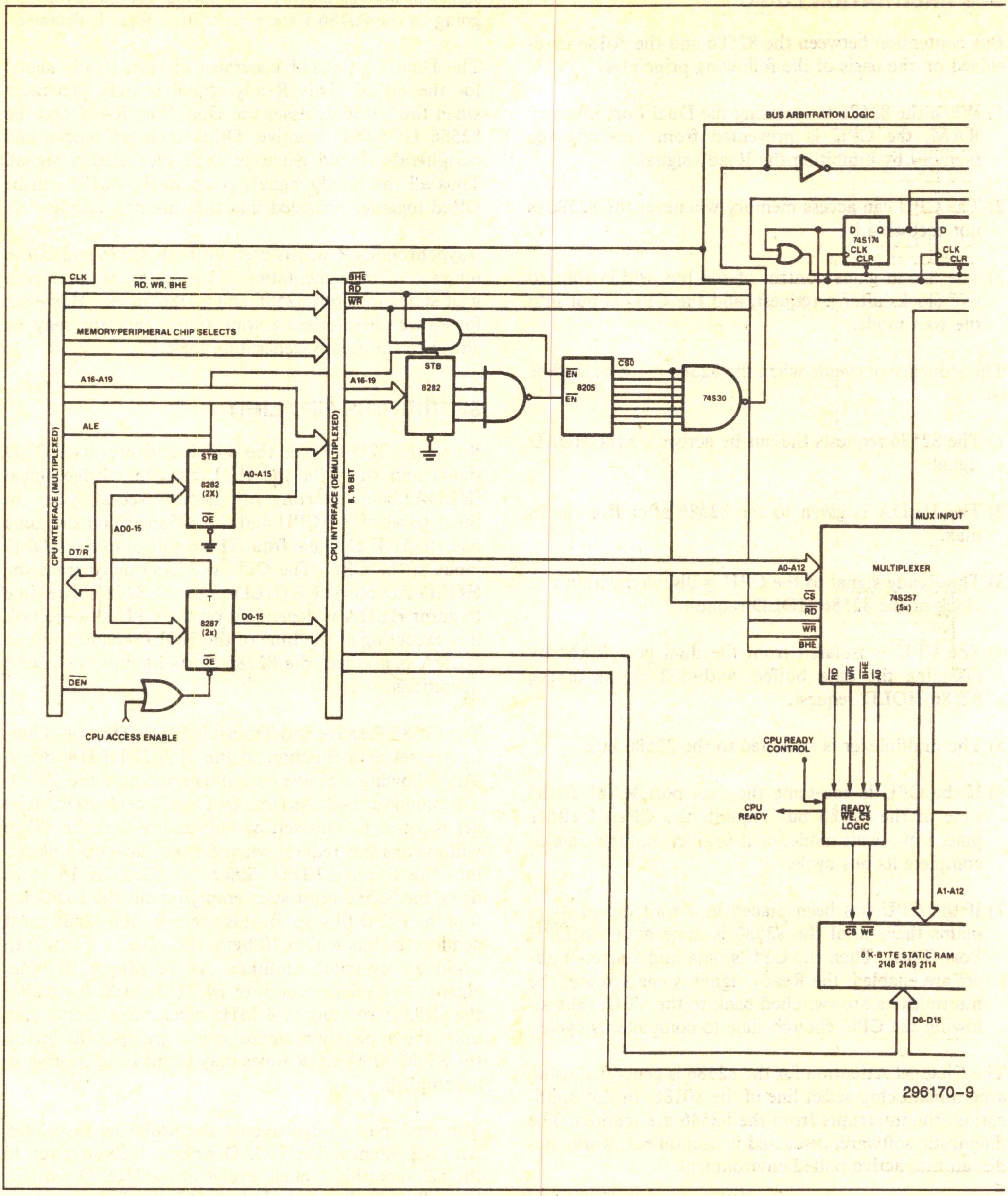


Figure 9. General Purpose CPU Interface

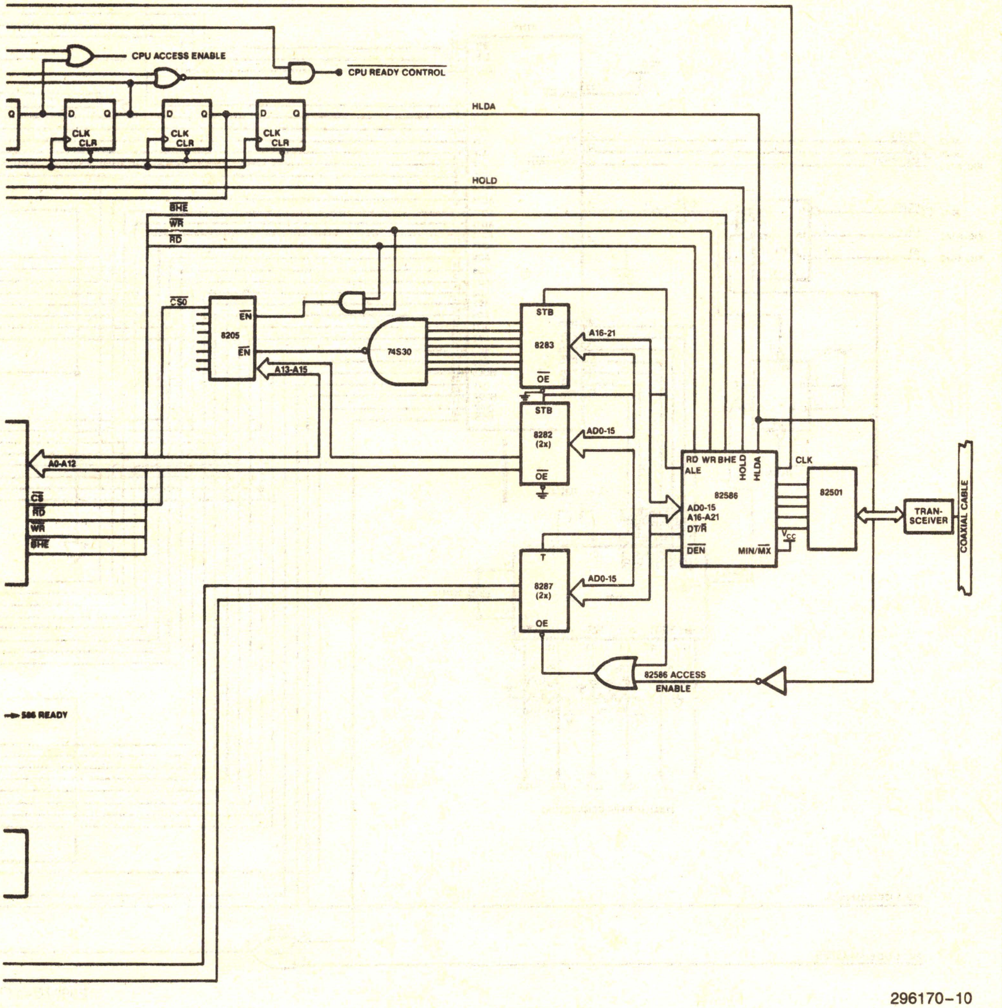


Figure 9. General Purpose CPU Interface (Continued)

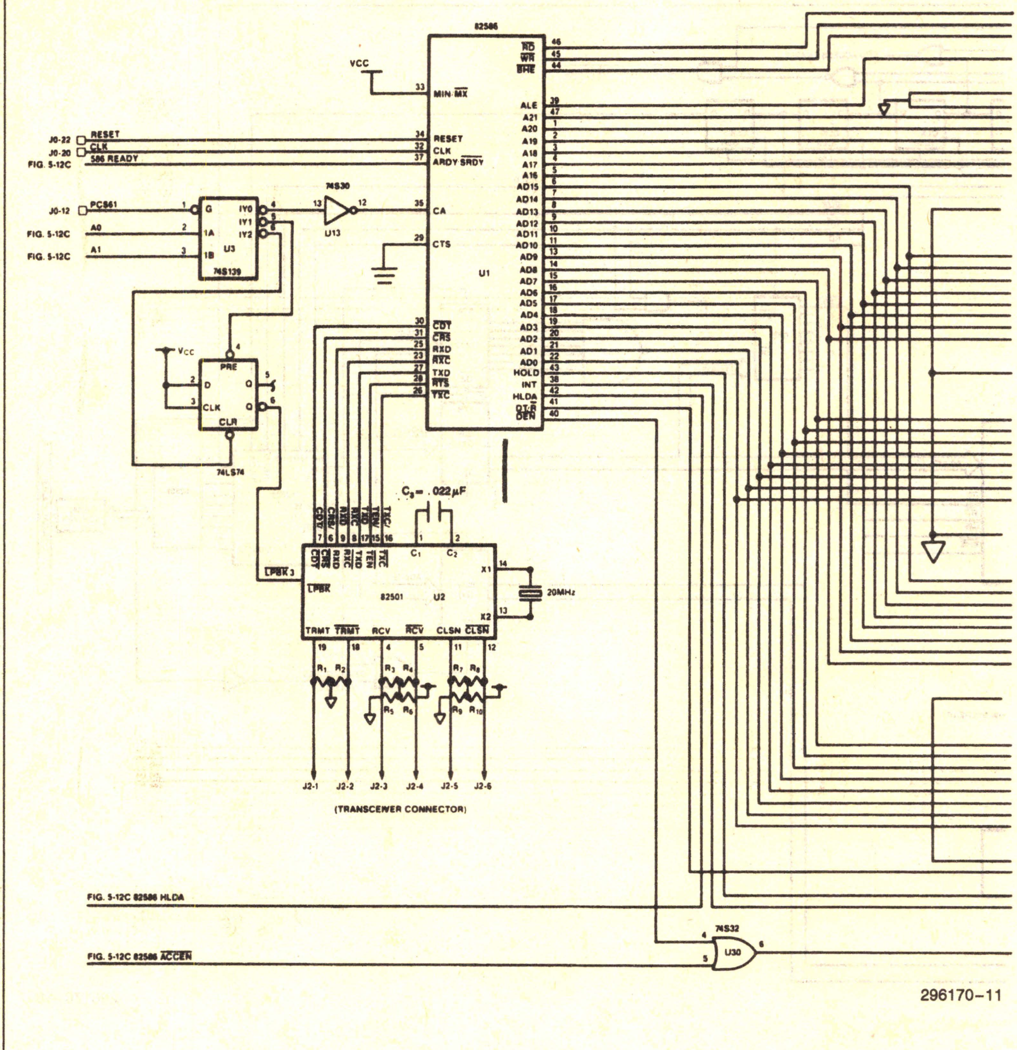


Figure 10A. 82586 Dual Port Design Schematics

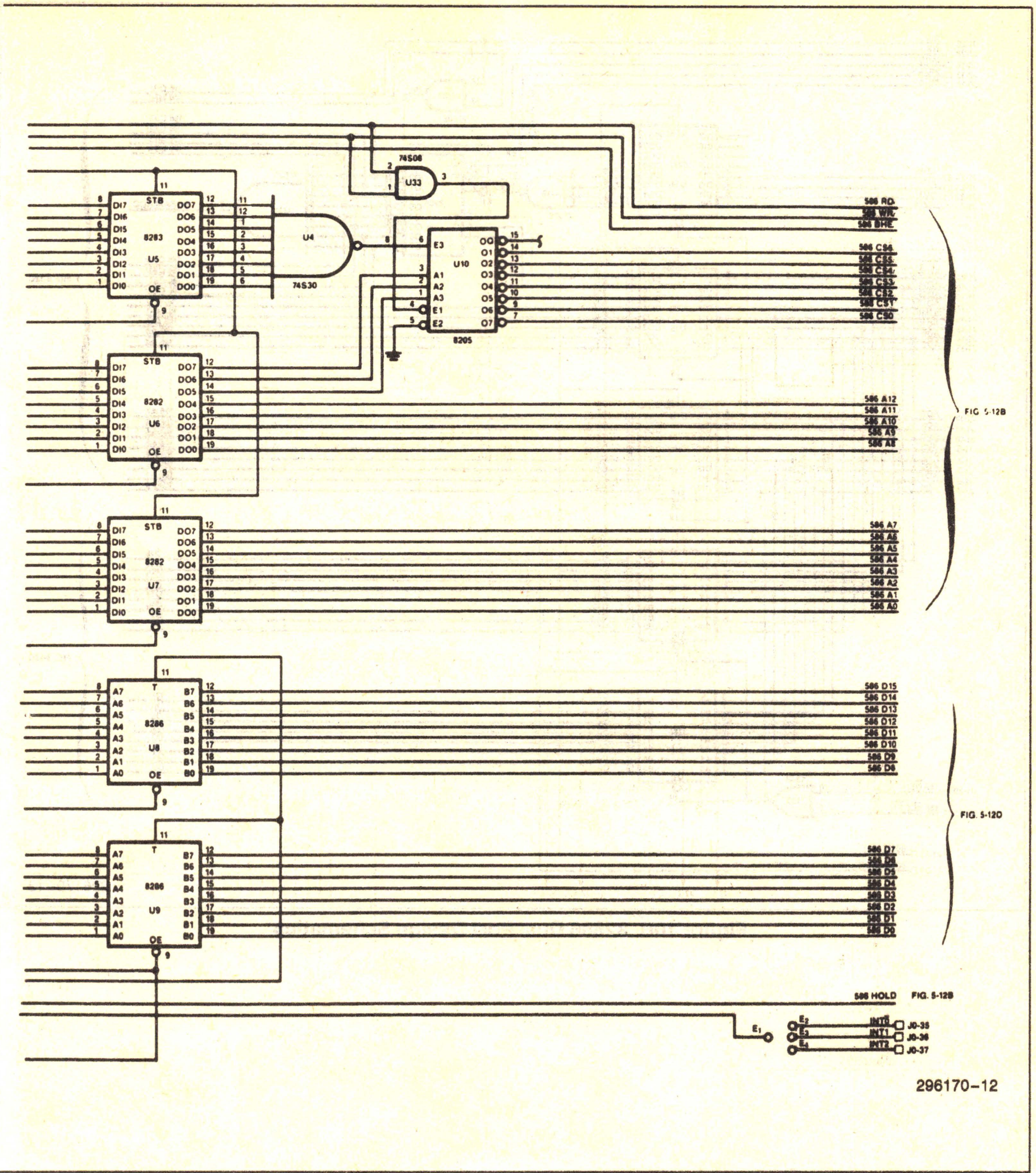


Figure 10A. 82586 Dual Port Design Schematics (Continued)

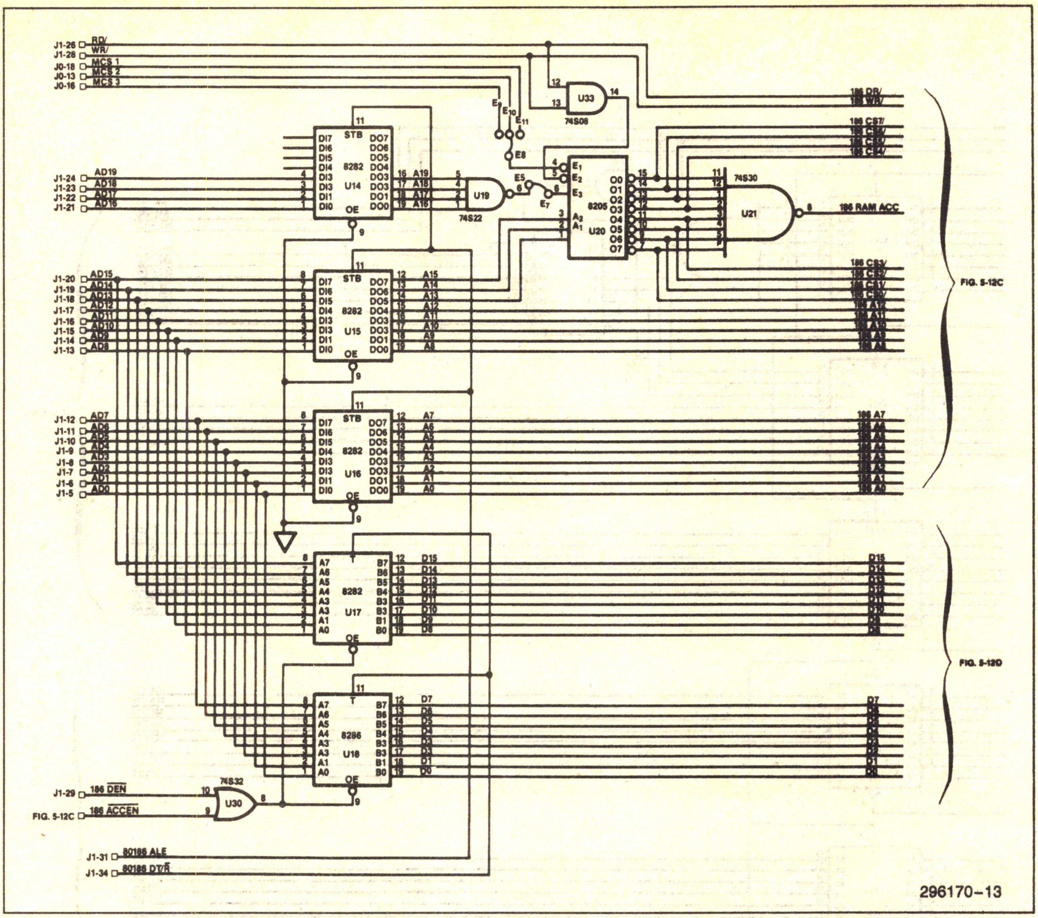


Figure 10B. 82586 Dual Port Design Schematics

Thus maximum bus efficiency is achieved when the 82586 acquires the bus with a full FIFO. A very small HOLD-HLDA delay will allow a high setting of the FIFO-Threshold. The HOLD-HLDA handshake between the 82586 and the CPU always results in a finite amount of wasted bus clock cycles due to HOLD and HOLD Acknowledge synchronization by the 82586 and the CPU.

In summary, the HOLD-HLDA handshake results in wasted bus clocks. In order to realize an efficient system, the 82586 must transfer the maximum number of bytes before giving up control of the bus. In the case of the 82586, these burst sizes should be 16 bytes (i.e. the FIFO size). The objective of the design should be to have 16 bytes in the FIFO when the 82586 gets the bus, so that it may empty out the FIFO in a single burst. A short HOLD-HLDA delay reduces the concern for DMA overruns; thus a high FIFO-Threshold setting is possible.

Another advantage of this design is that HLDA is always given to the 82586 between 4 and 5 clock cycles after a HOLD request. This provides more efficient bus utilization than with a direct microprocessor interface where HOLD and HLDA latency will vary much more than one clock cycle.

The discussion above centered on reception, however the same arguments are true for Transmission. When programming the FIFO-Threshold using the CONFIGURE command, the parameter programmed is the threshold point for the Transmit FIFO. The Receive FIFO-Threshold is sixteen's compliment (15-trigger parameter) of the value programmed.

Hardware schematics are shown in Figure 10A through 10D.

6.2 Application Software

The 82586 dual port hardware design was tested using an 80186 based board, running at 8 MHz. Two ribbon cables access the 80186 address, data, and control signals. These signals are brought to the 82586 board via connectors J0 and J1 (see Figure 10E). The 80186 board operated with an Intel iSBC 957B package monitor program. The 957B monitor was used to debug 82586 hardware and software. For more information on the 957B monitor, refer to the "iSBC 957B™ iAPX 86/88 User's Guide" (Intel order number 143979-002).

The 82586 diagnostic software allows interactive usage. The software was developed using an Intel MDS Series III and downloaded to the 82586 board through the serial link on the 80186 board.

The diagnostic software has the following capabilities.

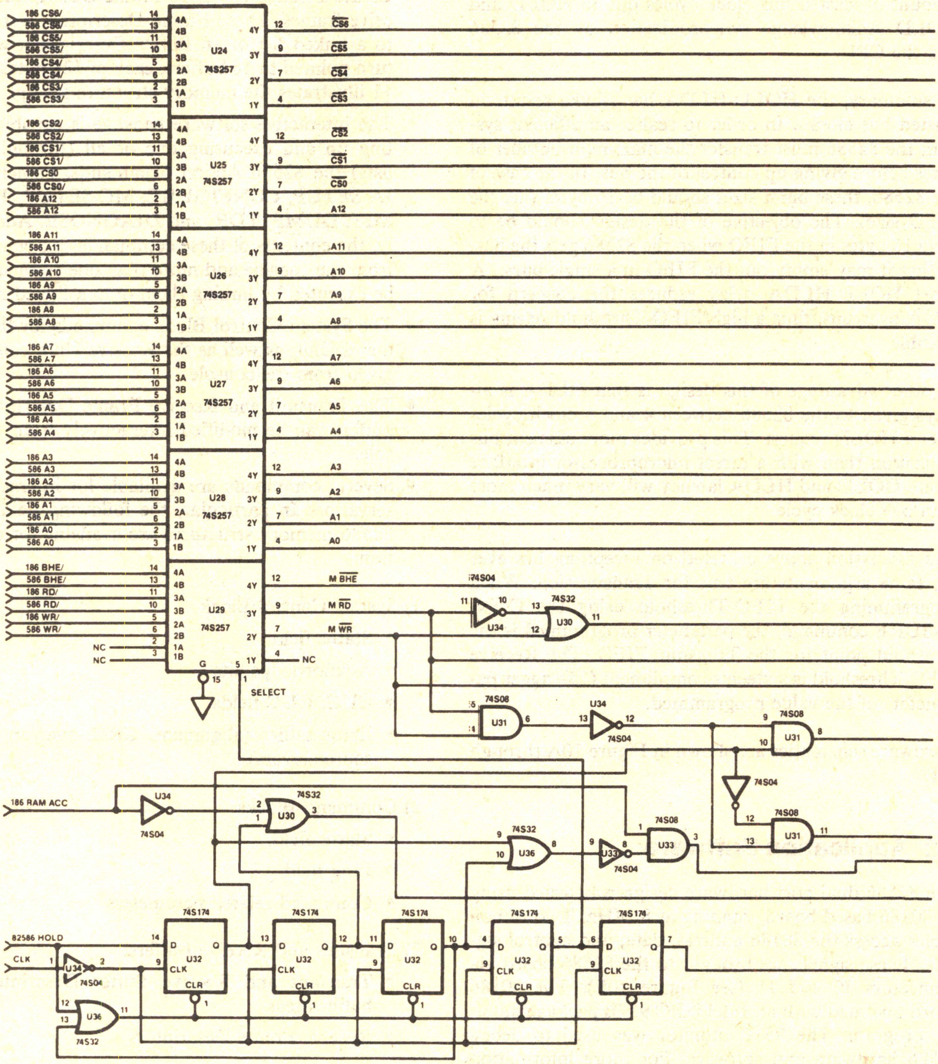
- Initialize the 82586 by setting up the System Control Pointer, Intermediate System Control Pointer and System Control Block.
- Create a linked list of 16 Frame Descriptors for receive frames. These Frame Descriptors were linked to a linked list of 16 Buffer Descriptors, which in turn pointed to 16 buffers, each of 128 bytes. Figure 11 illustrates the memory structure set up.
- The interactive software package is capable of setting up and executing one or all (through linked lists) the 82586 Action Commands, namely NOP, IA SETUP, CONFIGURE, MC SETUP, TRANSMIT, DUMP, TDR, and DIAGNOSE. Additionally, the contents of these commands can be modified from the console and more than one command may be executed by linking them onto a linked list.
- The System Control Block commands for the Command Unit, as well as the Receive Unit, can also be given from the console.
- The Transmit and Receive Frame Descriptors and buffers can be modified interactively from the console.
- Several commands are available for interactive observation. In particular, the following fields in the 82586 memory structures are available for observation:

1) System Control Block:
   - Status field
   - Acknowledge bits
   - CUC, RUC fields
   - Error tallies (alignment, CRC, overrun and resource errors)

2) Command Blocks:
   - Status field
   - Link field
   - Command related parameters

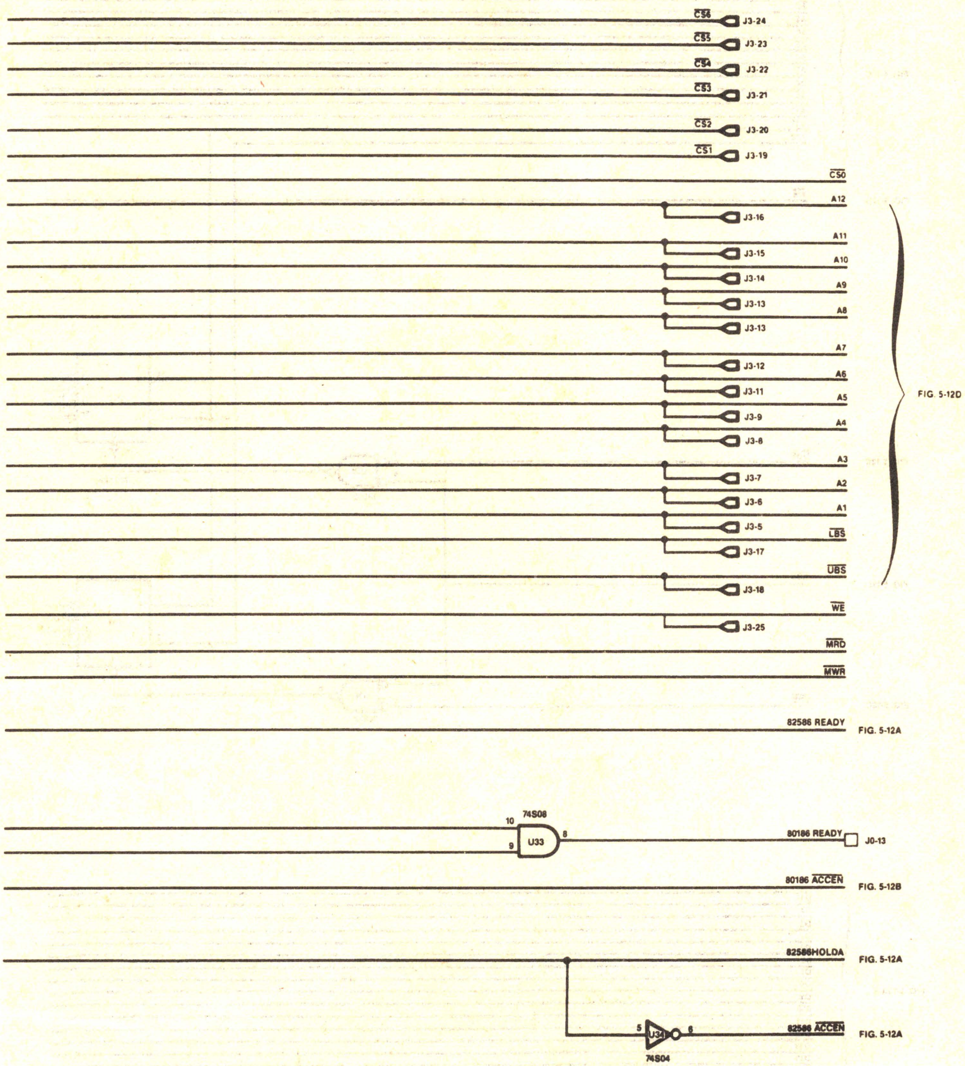
3) Transmit and Receive Buffers:
   - Transmit and Receive Buffer Descriptors and buffer areas
   - Receive Frame Descriptors

Table 6 contains the software listings for this application. Figure 12A through 12K illustrates the flow for the software modules.



296170-14

Figure 10C. 82586 Dual Port Design Schematics



296170-15

Figure 10C. 82586 Dual Port Design Schematics (Continued)

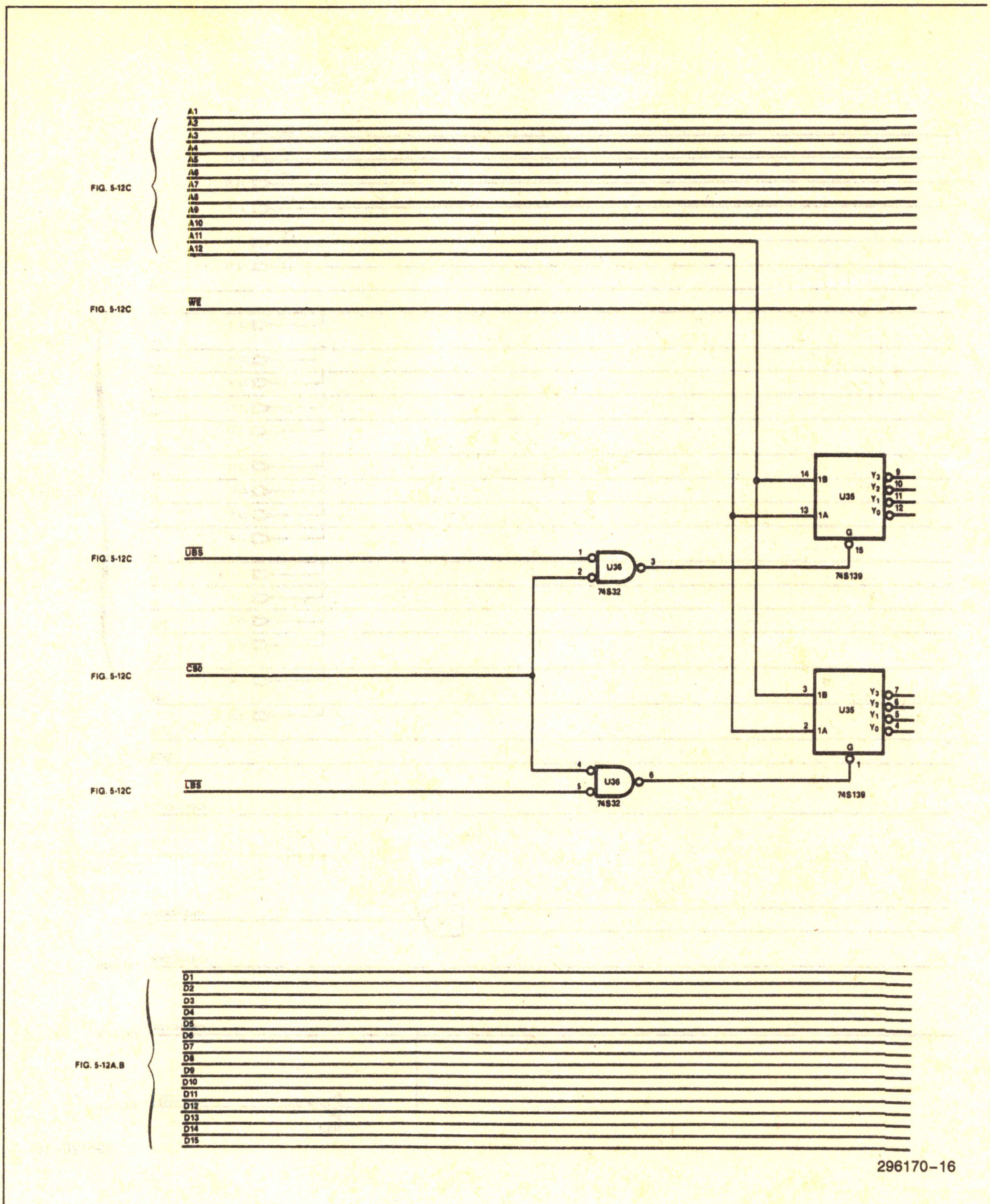
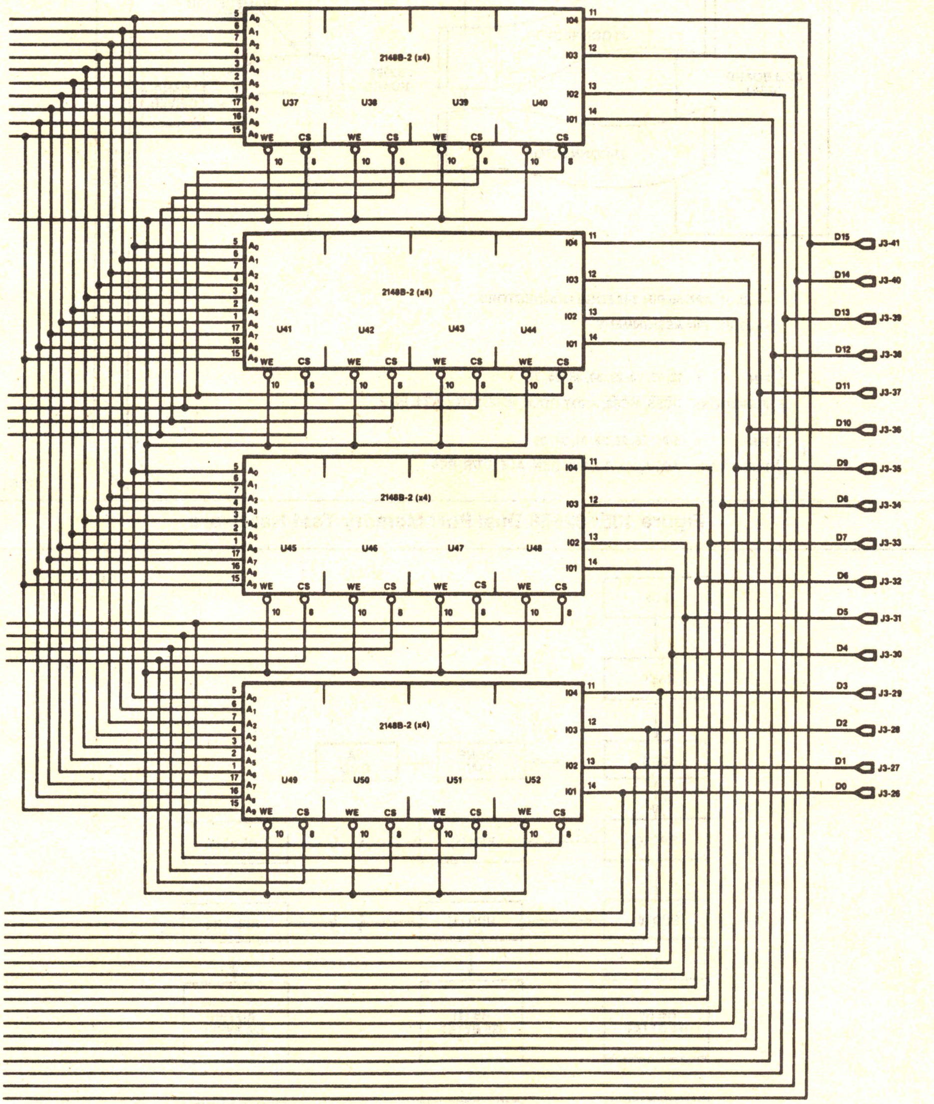


Figure 10D. 82586 Dual Port Design Schematics



296170-17

Figure 10D. 82586 Dual Port Design Schematics (Continued)

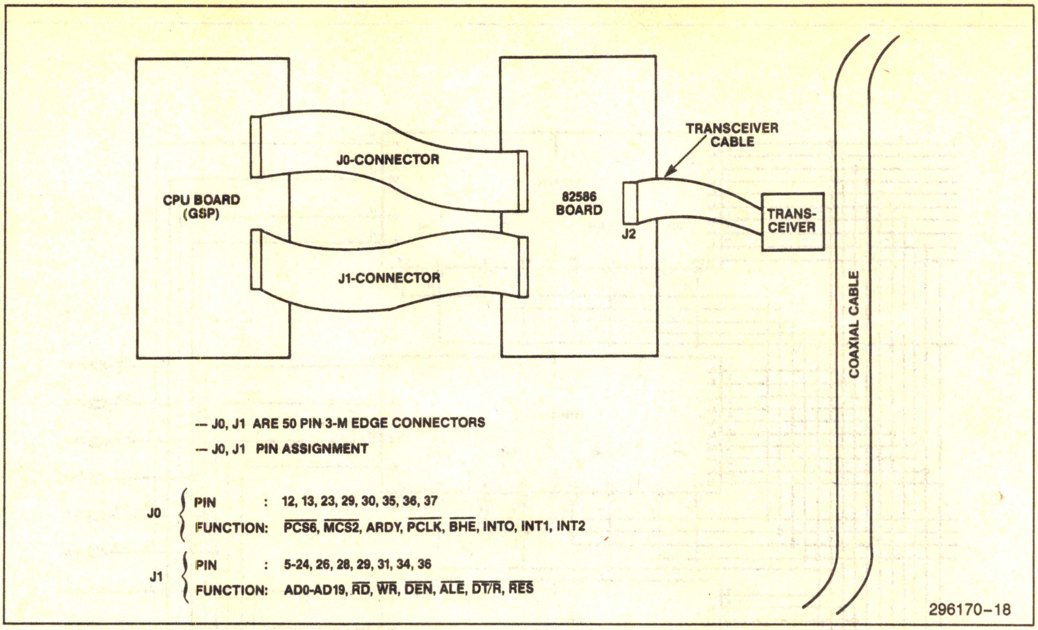


Figure 10E. 82586 Dual Port Memory Test Hardware

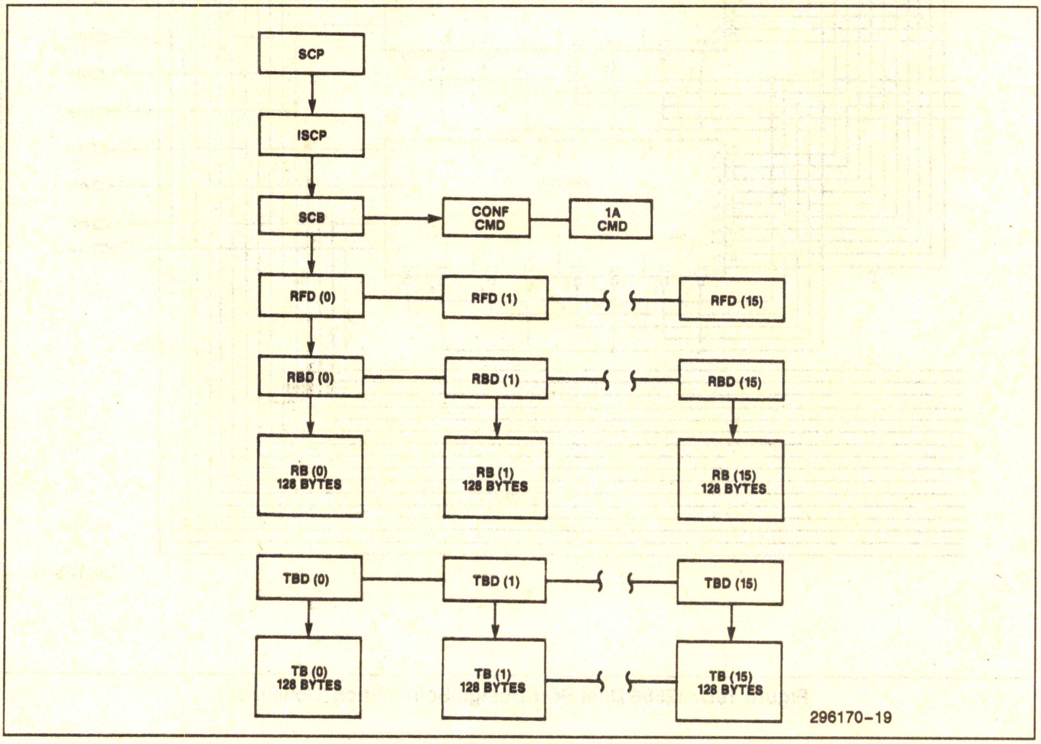


Figure 11. Memory Map for 82586

Table 6. Software Listings

```
/***********************************************************************
*                                                                      *
*                                                                      *
*                  82586 DUAL PORT MEMORY DESIGN                       *
*                                                                      *
*                  INTERACTIVE DIAGNOSTIC SOFTWARE                     *
*                                                                      *
*                  JAN. '84      SN/MH                                 *
*                                                                      *
*                                                                      *
***********************************************************************/


   1          DIAGNOSTICS$FOR$82586: DO;

   2    1     declare MAIN label public;

              /* Word Declare */

   3    1         declare FOREVER          literally 'WHILE 1';

              /* Constant Declare */

   4    1         declare CA$PTR               literally '700H',
                          LPBK$ON$PTR          literally '701H',
                          LPBK$OFF$PTR         literally '702H',
                          CMD$START            literally '0100H',
                          RCV$START            literally '0010H',
                          CMD$ABORT            literally '0400H',
                          RCV$ABORT            literally '0040H',
                          CMD$SUSPEND          literally '0300H',
                          RCV$SUSPEND          literally '0030H',
                          CMD$RESUME           literally '0200H',
                          RCV$RESUME           literally '0020H',
                          C$RES                literally '0080H',
                          CR                   literally '0DH',
                          LF                   literally '0AH',
                          BS                   literally '08H',
                          SP                   literally '20H',
                          DEL                  literally '07FH',
                          BEL                  literally '07H',
                          TRUE                 literally '0FFH',
                          FALSE                literally '0H',
                          TRUEW                literally '0FFFFH';


              /* Address Declare */

   5    1         declare SCP$OFFSET           word data (0FFF6H);
   6    1         declare ISCP$OFFSET          word data (0FFE0H);
   7    1         declare SCB$OFFSET           word data (0FFD0H);
   8    1         declare RX$OFFSET            word data (0E000H);
   9    1         declare TX$OFFSET            word data (0E800H);
  10    1         declare NOP$OFFSET           word data (0FF70H);
  11    1         declare IA$SETUP$OFFSET      word data (0FF76H);
  12    1         declare CONF$OFFSET          word data (0FF82H);
  13    1         declare MC$SETUP$OFFSET      word data (0FF94H);
  14    1         declare TRANSMIT$OFFSET      word data (0FFA2H);
  15    1         declare TDR$OFFSET           word data (0FFB2H);
  16    1         declare DUMP$STAT$OFFSET     word data (0FFBAH);
  17    1         declare DIAG$OFFSET          word data (0FFC2H);
  18    1         declare RFD$OFFSET           word data (0F200H);
  19    1         declare RBD$OFFSET           word data (0F000H);
  20    1         declare TBD$OFFSET           word data (0F100H);
  21    1         declare DS$BUFF$OFFSET       word data (0F400H);
  22    1         declare EX$TRANSMIT$OFFSET   word data (0F500H);


  23    1         declare MSG$PTR                              pointer,
                          MSG$BUF   based MSG$PTR   (5)        byte,
                          INT$PTR                              pointer,
```

296170-20

Table 6. Software Listings (Continued)

```
                    C$BUF      (160)                    byte,
                    ASCII$BUF    (2)                    byte,
                    INT$BUF                             byte,
                    NEXT$CMD$OFFSET                     word,
                    CURRENT$CMD$OFFSET                  word,
                    CMD$PTR                             pointer,
                    DAT based CMD$PTR                   byte,
                    MESSAGE$WORD                        word,
                    MESSAGE$BYTE                        byte,
                    MESSAGE$STATUS                      byte,
                    CMD$TOP$STATUS                      byte,
                    CMD$TOP$OFFSET                      word;

24   1        declare COMMON$BASE          selector     data (6000H);
25   1        declare SCB$BASE             word         data (6);


          /* System Configuration Pointer */

26   1        declare SCP$PTR              pointer;
27   1        declare SCP                  structure
                                           (SYSBUS              word,
                                            DUMMY               dword,
                                            ISCP$offset         word,
                                            iscp$base           word) at(6FFF6h);


          /* Intermediate System Control Pointer */
28   1        declare ISCP$PTR             pointer;
29   1        declare ISCP                 structure
                                           (BUSY                word,
                                            SCB$OFFSET          word,
                                            SCB$BASE1           word,
                                            SCB$BASE2           word) at(6FFE0H);


          /* System Control Block */

30   1        declare SCB$PTR              pointer;
31   1        declare SCB                  structure
                                           (STAT                word,
                                            CMD                 word,
                                            CBL$OFFSET          word,
                                            RFD$OFFSET          word,
                                            CRC$ERRS            word,
                                            ALN$ERRS            word,
                                            RSC$ERRS            word,
                                            OVRN$ERRS           word) at(6FFD0H);



          /* Receive Frame Descriptor */

32   1        declare RFD$PTR              pointer;
33   1        declare RFD   (10h)          structure
                                           (STAT                word,
                                            EL$S                word,
                                            LINK$ADDRESS        word,
                                            BD$PTR              word,
                                            DEST$ADDRESS (6)    byte,
                                            SOURCE$ADDRESS (6)  byte,
                                            TYPE$FIELD          word) at(6F200H);

          /* Receive Buffer Descriptor */

34   1        declare RBD$PTR              pointer;
35   1        declare RBD   (10H)          structure
                                            (ACT$COUNT          word,
                                             NEXT$BD$ADD        word,
                                             BUFF$OFFSET        word,
                                             BUFF$BASE          word,
                                             SIZE               word) at(6F000H);


          /* Transmit Buffer Descriptor */

36   1        declare TBD$PTR              pointer,
                      TBD   (10H)          structure
                                           (ACT$COUNT           word,
                                            NEXT$BD$ADD         word,
                                            BUFF$OFFSET         word,
                                            BUFF$BASE           word) at(6F100H);
```

296170-21

Table 6. Software Listings (Continued)

```
              /* NOP Command */

37    1            declare NOP$PTR              pointer;
38    1            declare NOP                  structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word) at(6FF70H);


              /* Individual Address Setup */

39    1            declare IA$SETUP$PTR         pointer,
                           IA$SETUP             structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 IA$ADDRESS (6)       byte) at(6FF76H);


              /* Configuration Command */

40    1            declare CONF$PTR             pointer,
                           CONF                 structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 BYTE$CNT             byte,
                                                 FIFO$LIM             byte,
                                                 SAV$BP$RDY           byte,
                                                 LPBK$PRELEN$ADDRLEN byte,
                                                 PRIORITY             byte,
                                                 IF$SPACE             byte,
                                                 SLOT$TIME            byte,
                                                 RETRY$NUM$SLT$TMH    byte,
                                                 MOD$CNTL             byte,
                                                 FILTER               byte,
                                                 MIN$FRAME$LEN        word) at(6FF82H);


              /* Multicust-ID Setup */

41    1            declare MC$SETUP$PTR         pointer,
                           MC$SETUP             structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 MC$CNT               word,
                                                 MC$ID (6)            byte) at(6FF94H);


              /* Transmit Command */

42    1            declare TRANSMIT$PTR         pointer,
                           TRANSMIT             structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 BUF$DESC$PTR         word,
                                                 DEST$ADDRESS (6)     byte,
                                                 TYPE$FIELD           word) at(6FFA2H);


              /* Time Domain Reflectmeter Test */

43    1            declare TDR$PTR              pointer,
                           TDR                  structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 TIME                 word) at(6FFB2H);


              /* Dump Status */

44    1            declare DUMP$STAT$PTR        pointer,
                           DUMP$STAT            structure
                                                (STAT                 word,
                                                 CMD                  word,
                                                 LINK$ADDRESS         word,
                                                 BUF$PTR              word) at(6FFBAH);
```

296170-22

Table 6. Software Listings (Continued)

```
              /* Diag */

45    1           declare DIAG$PTR                 pointer,
                          DIAG                     structure
                                                    (STAT                word,
                                                     CMD                 word,
                                                     LINK$ADDRESS        word) at(6FFC2H);


              /* Receive Buffer */

46    1           declare RXb$PTR pointer,
                          RX   (10H)               structure
                                                   (BUF(80H) byte) at(6E000H);


              /* Transmit Buffer */

47    1           declare TXB$PTR                  pointer,
                          TX   (10H)               structure
                                                   (BUF(80H) byte) at(6E800H);

              /* Extra Transmit Command */

48    1           declare EX$TRANSMIT$PTR           pointer,
49    1           declare EX$TRANSMIT(10)           structure
                                                   (STAT                 word,
                                                    CMD                  word,
                                                    LINK$ADDRESS         word,
                                                    BD$PTR               word,
                                                    DEST$ADDRESS(6)      byte,
                                                    TYPE                 word) at(6F500H),

              /* Channel Attention */

50    1           CA: procedure;
51    2               call time(1);
52    2               output(CA$PTR) = 1;
53    2           end CA;


              /* Acknowledge routine */

54    1           ACK: procedure;

55    2               SCB.CMD = SCB.STAT and 0F000H;
56    2               call CA;

57    2           end ACK;


              /* Command Block Initialization Procedure */


              /* NOP COMMAND */

58    1           NOP$CMD: procedure;


59    2               NOP.STAT = 0;
60    2               NOP.CMD = 0A000H;
61    2               NOP.LINK$ADDRESS = 0FFFFH;
62    2               SCB.CMD = CMD$START;

63    2           end NOP$CMD;

              /* INDIVIDUAL ADDRESS SET UP COMMAND */

64    1           IA$SETUP$CMD: procedure;


65    2               IA$SETUP.STAT = 0;
66    2               IA$SETUP.CMD   = 0A001H,
67    2               IA$SETUP.LINK$ADDRESS = 0FFFFH;
68    2               IA$SETUP.IA$ADDRESS (0) = 0;      /* INDIVIDUAL ADDRESS  6BYTE */
69    2               IA$SETUP.IA$ADDRESS (1) = 0AAH;
70    2               IA$SETUP.IA$ADDRESS (2) = 0;
71    2               IA$SETUP.IA$ADDRESS (3) = 12H,
72    2               IA$SETUP.IA$ADDRESS (4) = 34H;
73    2               IA$SETUP.IA$ADDRESS (5) = 56H;
74    2               SCB.CMD = CMD$START;
```

296170-23

Table 6. Software Listings (Continued)

```
 75   2          end IA$SETUP$CMD;

              /* CONFIGURATION BLOCK INITIALIZATION */

 76   1          CONF$CMD: procedure;

 77   2              CONF.STAT = 0;                        /* Configuration command status,
                                                              it should be set to 0. */
 78   2              CONF.CMD = 0A002H;                    /* Configuration command word */
 79   2              CONF.LINK$ADDRESS = 0FFFFH;           /* Next command link address */
 80   2              CONF.BYTE$CNT = 12;                   /* Configuration parameter byte
                                                              count */
 81   2              CONF.FIFO$LIM = 8;                    /* 8-bytes FIFO limit */
 82   2              CONF.SAV$BP$RDY = 0;
 83   2              CONF.LPBK$PRELEN$ADDRLEN = 26H;       /* 6-bytes address length,
                                                              8-bytes preamble length */
 84   2              CONF.PRIORITY = 0;
 85   2              CONF.IF$SPACE = 96;                   /* Interframe spacing 9.6 micro sec */
 86   2              CONF.SLOT$TIME = 0;
 87   2              CONF.RETRY$NUM$SLT$TMH = 0F2H;        /* Retry number =15, slot time 51.2
                                                              micro sec */
 88   2              CONF.MOD$CNTL = 1;                    /* Promiscuous mode */
 89   2              CONF.FILTER = 0;                      /* CRS and CDT filters not used  */
 90   2              CONF.MIN$FRAME$LEN = 64;              /* Minimum frame length 64-bytes */
 91   2              SCB.CMD = CMD$START;

 92   2          end CONF$CMD;

              /* MULTICAST-ID SETUP COMMAND */

 93   1          MC$SETUP$CMD: procedure;

 94   2              MC$SETUP.STAT = 0;                     /* it should be 0. */
 95   2              MC$SETUP.CMD = 0A003H;                 /* Mulitcast command */
 96   2              MC$SETUP.LINK$ADDRESS = 0FFFFH;
 97   2              MC$SETUP.MC$CNT = 6H;
 98   2              MC$SETUP.MC$ID (0) = 0FFH;             /* Mulitcast IDs */
 99   2              MC$SETUP.MC$ID (1) = 0FFH;
100   2              MC$SETUP.MC$ID (2) = 12H;
101   2              MC$SETUP.MC$ID (3) = 34H;
102   2              MC$SETUP.MC$ID (4) = 56H;
103   2              MC$SETUP.MC$ID (5) = 78H;
104   2              SCB.CMD = CMD$START;
105   2          end MC$SETUP$CMD;

              /* TRANSMIT COMMAND */

106   1          TX$CMD: procedure;

107   2              TRANSMIT.STAT = 0,
108   2              TRANSMIT.CMD   = 0A004H,
109   2              TRANSMIT.LINK$ADDRESS =0FFFFH,
110   2              TRANSMIT.BUF$DESC$PTR = TBD$OFFSET;
111   2              TRANSMIT.DEST$ADDRESS (0) = 00;    /* TRANSMIT DESTINATION ADDRESS */
112   2              TRANSMIT.DEST$ADDRESS (1) = 0AAH;  /* 6BYTES                       */
113   2              TRANSMIT.DEST$ADDRESS (2) = 00,
114   2              TRANSMIT.DEST$ADDRESS (3) = 00;
115   2              TRANSMIT.DEST$ADDRESS (4) = 0AH;
116   2              TRANSMIT.DEST$ADDRESS (5) = 8EH;
117   2              TRANSMIT.TYPE$FIELD = 0;
118   2              SCB.CMD = CMD$START;

119   2          end TX$CMD;

              /* EXTRA TRANSMIT COMMAND */
120   1          EX$TX$CMD: procedure;
121   2              declare I                     byte,
122   2              do I = 0 to 9;
123   3                  EX$TRANSMIT(I).STAT = 0;
124   3                  EX$TRANSMIT(I).CMD   = 0A004H;
125   3                  EX$TRANSMIT(I).LINK$ADDRESS = 0FFFFH;
126   3                  EX$TRANSMIT(I).BD$PTR = TBD$OFFSET;
127   3                  EX$TRANSMIT(I).DEST$ADDRESS(0) = 00;
128   3                  EX$TRANSMIT(I).DEST$ADDRESS(1) = 0AAH;
129   3                  EX$TRANSMIT(I).DEST$ADDRESS(2) = 00;
130   3                  EX$TRANSMIT(I).DEST$ADDRESS(3) = 00;
131   3                  EX$TRANSMIT(I).DEST$ADDRESS(4) = 0AH;
132   3                  EX$TRANSMIT(I).DEST$ADDRESS(5) = 8EH;
133   3                  EX$TRANSMIT(I).TYPE = I;
134   3              end;
135   2              SCB.CMD = CMD$START;
136   2          end EX$TX$CMD;
```

296170-24

Table 6. Software Listings (Continued)

```
                   /* TIME DOMAIN REFLECTOMETER TEST COMMAND */

137    1               TDR$CMD: procedure;

138    2                   TDR.STAT = 0;
139    2                   TDR.CMD = 0A005H;
140    2                   TDR.LINK$ADDRESS = 0FFFFH;
141    2                   TDR.TIME = 0;
142    2                   SCB.CMD = CMD$START;

143    2               end TDR$CMD;


                   /* DUMP STATUS COMMAND */

144    1               DUMP$STAT$CMD: procedure;

145    2                   DUMP$STAT.STAT = 0;
146    2                   DUMP$STAT.CMD = 0A006H;
147    2                   DUMP$STAT.LINK$ADDRESS = 0FFFFH;
148    2                   DUMP$STAT.BUF$PTR = DS$BUFF$OFFSET;      /* DUMP STATUS RESULT OFFSET */
149    2                   SCB.CMD = CMD$START;
150    2               end DUMP$STAT$CMD;


                   /* Diagnose Command */

151    1               DIAG$CMD: procedure;

152    2                   DIAG.STAT = 0;
153    2                   DIAG.CMD  = 0A007H;
154    2                   DIAG.LINK$ADDRESS = 0FFFFH;
155    2                   SCB.CMD = CMD$START;

156    2               end DIAG$CMD;



                   /* Receive Frame Descriptor set up */

157    1               INIT$RFD: procedure;

158    2                   declare I                          word;
159    2                   call setb(0,@RFD(0).STAT,352);
160    2                   RFD (0).STAT = 0;
161    2                   RFD (0).LINK$ADDRESS = RFD$OFFSET + offset$of(@RFD (1));
162    2                   RFD (0).BD$PTR = RBD$OFFSET + offset$of(@RBD(0));

163    2                   do I = 1 to 14;
164    3                       RFD (I).STAT = 0;
165    3                       RFD (I).LINK$ADDRESS = RFD$OFFSET + offset$of(@RFD(I + 1));
166    3                       RFD (I).BD$PTR = 0FFFFH;
167    3                   end;
168    2                   RFD (15).STAT = 0;
169    2                   RFD (15).LINK$ADDRESS = 0FFFFH;
170    2                   RFD (15).BD$PTR = 0FFFFH;
171    2                   RFD (15).EL$S = 8000H;

172    2               end INIT$RFD;

                   /* Receive Buffer Descriptor set up

                           ------------       ------------                         --------------
                   --->    | RBD(0)   | --->  | RBD(1)   | ---> ..........--->     | RBD(15)    |
                           ------------       ------------                         --------------
                           | EL = 0   |       | EL = 0   |                         | EL = 1     |
                           ------------       ------------                         --------------
                           | NBD=RB(0)|       | NBD=RB(1)|                         | NBD= FFFFH |
                           ------------       ------------                         --------------

                                                                                                */

173    1               INIT$RBD: procedure;

174    2                   declare I          word;

175    2               do I = 0 TO 14;
176    3                   RBD(I).ACT$COUNT = 0;
177    3                   RBD(I).NEXT$BD$ADD = RBD$OFFSET + offset$of(@RBD(I+1));
```

296170-25

Table 6. Software Listings (Continued)

```
178   3                 RBD(I).SIZE = 80H;
179   3         end;
180   2                 RBD(15).ACT$COUNT = 0;
181   2                 RBD(15).NEXT$BD$ADD = OFFFFH;
182   2                 RBD(15).SIZE = 8080H;
183   2         end INIT$RBD;

            /* Receive Buffer set up */

184   1         INIT$RB: procedure;

185   2             declare I               word;
186   2             declare VALUE           byte;
187   2             declare COUNT           word;

188   2             VALUE = 0;
189   2             COUNT = 800H;
190   2             call setb(VALUE,@RX,COUNT); /* Full up receive buffer with 00h */

191   2             do I = 0 TO 15;
192   3                 RBD(I).BUFF$BASE = SCB$BASE;
193   3                 RBD(I).BUFF$OFFSET = RX$OFFSET + offset$of(@RX(I));
194   3             end;

195   2         end INIT$RB;


            /* Transmit Buffer Descriptor setup
                                1) Setup actual count for Transmit Buffer Descriptor,
                                   and one frame is transmitted by one Transmit
                                   command.

                                2) Setup next buffer descriptor address for each TBD
                                   in order to link with next TBD.

                 ------------       ------------                      -------------
            ---> | TBD (0)    | ---> | TBD (1)    | ---> ........ ---> | TBD (15)    |
                 ------------       ------------                      -------------
                 | EL = 0     |      | EL = 0     |                     | EL = 1      |
                 ------------       ------------                      -------------
                 | NBD=TB(0)  |      | NBD=TB(1)  |                     | NBD= FFFFH  |
                 ------------       ------------                      -------------
                                                                                    */

196   1         INIT$TBD: procedure;

197   2             declare I           word;

198   2             do I = 0 TO 14;
199   3                 TBD(I).ACT$COUNT = 80H;
200   3                 TBD(I).NEXT$BD$ADD = TBD$OFFSET + offset$of(@tbd(i + 1));
201   3             end;
202   2             TBD(15).ACT$COUNT = 8080H;
203   2             TBD(15).NEXT$BD$ADD = OFFFFH;
204   2         end INIT$TBD;


            /* Transmit buffer set up

                                1. Set up TBD
                                        - each TBD has own transmit buffer
                                2. Set up byte count of each buffer
                                        - 80h bytes
                                3. Set up buffer data

               */

205   1         INIT$TB: procedure;

206   2             declare I               byte,
                            J               byte,
                            NEW$VALUE       byte,
                            DEST$ADDR       pointer,
                            BYT$CNT         byte,
                            TXB$PTR         pointer;

207   2             do I = 0 TO 15;
208   3                 TBD(I).BUFF$BASE = SCB$BASE;
209   3                 TBD(I).BUFF$OFFSET = TX$OFFSET + offset$of(@TX(I));
210   3             end;
```

296170-26

Table 6. Software Listings (Continued)

```
2               J = 0;
2               do I = 0 TO 15;
3               byt$cnt = 80h;
3               NEW$VALUE = J;
3               DEST$ADDR = @TX(I);
3               call setb(NEW$VALUE, DEST$ADDR, BYT$CNT); /* Set up buffer data for
                                                             transmission          */

3               J = J + 01H;
3               end;

2           end INIT$TB;



    /* System I/O routine */

    /* Console input routine --- returns to the AL register an ASCII character
                                 received from the console device.  The AX, CX,
                                 and DX registers and the CPU condition codes
                                 are affected by this operation       */

1           CI: procedure byte external;
2               end CI;


    /* Console output routine --- transfers a character from the low-order byte
                                  of the word on the top of the stack to the
                                  console device.  The AX, CX, and DX registers
                                  and the CPU condition codes are affected by
                                  this operation.                             */
1           CO: procedure (CHARACTER) external;
2               declare CHARACTER          byte;
2           end CO;

    /* Exit routine --- causes a jump to the iAPX 86, 88 monitor command level.
                        EXIT does not close any files.                   */

1           EXIT: procedure external;
2           end EXIT;

    /* Carriage  Return ---- Parameter  : none */

1           CR$LF:  procedure;
2                   call CO(0DH);
2                   call CO(0AH);
2           end     CR$LF;


    /* Out string routine --- transfers string to console device.   Null code
                              (0h) found in the string, terminates the string
                              transfer.                                        */

1           OUTS: procedure;
2               declare CHAR                byte;
2               declare I                   byte;

2               I = 0;
2               CHAR = 0FFH;
2               do while CHAR <> 0;
3                       CHAR = MSG$BUF(I);
3                       call CO (CHAR);
3                       I = I + 1;
3               end;
2           end OUTS;

    /* Input string routine --- transfers string from console device.  The input
                                code of carriage return (CR, 0DH) or line feed (LF,
                                0AH) causes termination of this routine.
                                This routine supports rubout and back space also.  */
```

296170-27

Table 6. Software Listings (Continued)

```
INS: procedure;
    declare CHAR                       byte;
    declare I                          byte;

    CHAR = 10H;
    do I = 0 to 159;
        C$BUF(I) = CHAR;
    end;

    I = 0;
    CHAR = 10H;
    do while ((CHAR <> CR) and (CHAR <> LF) and (I <> 158));
        CHAR = CI and 7FH;
        if (char = del) or (CHAR = BS)
        then do;
            if I > 0
            then do;
                I = I - 1;
                call CO(BS);
                call CO(SP);
                call CO(BS);
            end;
            else call CO(BEL);
        end;
        else if CHAR >= SP
        then  do;
            call CO(CHAR);
            if ((CHAR >= 'a') and (CHAR <= 'z'))
            then C$BUF(I) = CHAR - 20H;
            else C$BUF(I) = CHAR;
            I = I + 1;
        end;
        else if ((CHAR = CR) or (CHAR = LF))
        then do;
            C$BUF(I) = CR;
            C$BUF(I + 1) = LF;
            I = I + 2;
        end;
        else  call CO(BEL);
    end;
end INS;


IN$YES: procedure byte;

    declare CHAR                byte;
    CHAR = 0;
    do while ((CHAR <> 'Y' or CHAR <> 'y')
        and (CHAR <> 'N' or CHAR <> 'n'));

        CHAR = CI and 7FH;
        call CO(CHAR);
        if (CHAR = 'Y' or CHAR = 'y') then
            return TRUE;
        else if (CHAR = 'N' or CHAR = 'n') then
            return FALSE;
        else do;
            MSG$PTR = @(0DH,0AH,'** KEY IN Y or N',0DH,0AH,2AH,0);
            call OUTS;
        end;
    end;
end IN$YES;

/* Software chip reset

                        Name:       CHIP$RESET
                        Input:      None
                        Output:     None
                        Function:   This routine performs 82586 chip reset
                                    by software.  It is same as Hardware
                                    RESET.                                 */

CHIP$RESET: procedure;
    SCB.CMD = 80H;
    call CA;
end CHIP$RESET;
```

296170–28

Table 6. Software Listings (Continued)

```
              /* System Initialize */

301   1           INIT$SYS: procedure;

302   2           call setb(0,@rx,2000h);
303   2           CMD$TOP$STATUS = FALSE;

              /* Pointer Declare */

304   2                SCP$PTR = build$ptr(COMMON$BASE, SCP$OFFSET);
305   2                ISCP$PTR = build$ptr(COMMON$BASE, ISCP$OFFSET);
306   2                SCB$PTR = build$ptr(COMMON$BASE, SCB$OFFSET);
307   2                NOP$PTR = build$ptr(COMMON$BASE, NOP$OFFSET);
308   2                IA$SETUP$PTR = build$ptr(COMMON$BASE, IA$SETUP$OFFSET);
309   2                CONF$PTR = build$ptr(COMMON$BASE, CONF$OFFSET);
310   2                MC$SETUP$PTR = build$ptr(COMMON$BASE, MC$SETUP$OFFSET);
311   2                TRANSMIT$PTR = build$ptr(COMMON$BASE, TRANSMIT$OFFSET);
312   2                TDR$PTR = build$ptr(COMMON$BASE, TDR$OFFSET);
313   2                DUMP$STAT$PTR = build$ptr(COMMON$BASE, DUMP$STAT$OFFSET);
314   2                DIAG$PTR = build$ptr(COMMON$BASE, DIAG$OFFSET);
315   2                RFD$PTR = build$ptr(COMMON$BASE, RFD$OFFSET);
316   2                RBD$PTR = build$ptr(COMMON$BASE, RBD$OFFSET);
317   2                RXB$PTR = build$ptr(COMMON$BASE, RX$OFFSET);
318   2                TBD$PTR = build$ptr(COMMON$BASE, TBD$OFFSET);
319   2                TXB$PTR = build$ptr(COMMON$BASE, TX$OFFSET);
320   2                EX$TRANSMIT$PTR = build$ptr(COMMON$BASE, EX$TRANSMIT$OFFSET);

              /* SCP initialization */

321   2                SCP.SYSBUS = 0H;              /* System bus width - 16 bits. */
322   2                SCP.ISCP$BASE = SCB$BASE;   /* ISCP start address 24 - bits */
323   2                SCP.ISCP$OFFSET = ISCP$OFFSET;

              /* ISCP initialization */

324   2                ISCP.BUSY = 01H;              /* This words set by CPU to 01H,
                                                         82586 clears it. */
325   2                ISCP.SCB$OFFSET = offset$of(SCB$PTR);   /* SCB start address offset */
326   2                ISCP.SCB$BASE1 = 0H;          /* Lower word of SCB base address. */
327   2                ISCP.SCB$BASE2 = 06H;         /* MS-byte of SCB base address. */

              /* SCB initalization */

328   2                SCB.STAT = 0H;                /* SCB STATUS word, it should be set to 0. */
329   2                SCB.CMD  = 0H;                /* SCB Command word */
330   2                SCB.CBL$OFFSET = CONF$OFFSET;
                                                    /* Command block list offset, the address
                                                       of command block list is the summation
                                                       of SCB BASE  and OFFSET value. */
331   2                SCB.RFD$OFFSET = offset$of(RFD$PTR); /* Receive frame area offset */
332   2                SCB.CRC$ERRS = 0;             /* CRC Error counter */
333   2                SCB.ALN$ERRS = 0;             /* Alignment error counter */
334   2                SCB.RSC$ERRS = 0;             /* No resource error counter (# of frames
                                                       were lost by no resource errors) */
335   2                SCB.OVRN$ERRS = 0;            /* DMA OVER-RUN error counter */
336   2                call CA;                      /* NOP command execution */
337   2                call CHIP$RESET;              /* Chip reset */
338   2                call ACK;                     /* Acknolidge for Chip reset */
339   2                call CONF$CMD;                /* Set configuration command parameter */
340   2                CONF.CMD = 0002H;             /* EL = 0; S = 0, I = 0, CMD = CONF */
341   2                CONF.LINK$ADDRESS = IA$SETUP$OFFSET; /* IA setup cmd for next cmd block */
342   2                call IA$SETUP$CMD;            /* IA setup cmd parameter set */
343   2                IA$SETUP.CMD = 0A001H;        /* IA command set */
344   2                NEXT$CMD$OFFSET = OFFFFH;     /* Next command offset for last cmd block */
345   2                call CA;                      /* Execute them */
346   2           end INIT$SYS;

              /* Message for HELP command */

347   1           COMMAND$MSG: procedure;

348   2                call CR$LF;
349   2                MSG$PTR = @('NOP ------------------- NOP
                             INDIVIDUAL ADDR SETUP - IA ', ODH, OAH,
                             'CONFIGURATION -------- CONF
                             MULTICAST-ID ---------- MC ', ODH, OAH, 0);
350   2                call OUTS;
351   2                MSG$PTR = @('TRANSMIT -------------- TX
                             TDR ------------------- ,DR', ODH, OAH,
                             'DUMP STATUS ----------- DS
                             DIAGNOSE -------------- DIAG', ODH, OAH, 0);
```

296170-29

Table 6. Software Listings (Continued)

```
352   2           call OUTS;

353   2       end COMMAND$MSG;


354   1       BD$MSG: procedure;
355   2           MSG$PTR = @('RX FRAME DESCRIPTOR --- RFD',ODH, OAH,
                        'RX BUFFER DESCRIPTOR -- RBD
                        RX BUFFER -------------- RB',ODH,OAH,O);
356   2           call OUTS;
357   2           MSG$PTR = @('TX BUFFER DESCRIPTOR --- TBD
                        TX BUFFER -------------- TB',ODH,OAH,O);
358   2       end BD$MSG;

     /* Message for illigal input */

359   1       MSG$ILL$CMD: procedure;
360   2               MSG$PTR = @('      <---- ILLEGAL INPUT', ODH, OAH, O);
361   2               call OUTS;
362   2       end MSG$ILL$CMD;



     /* Integer to ASCII code conversion

                        Name:       INT$TO$ASCII
                        Input:      INT$BUF       Integer buffer (1 byte);
                        Output:     ASCII$BUF     ASCII buffer (2 bytes);
                        Function:   INTEGER TO ASCII conversion */


363   1       INT$TO$ASCII: procedure;

364   2           declare TEMP$CHAR       byte;

365   2               TEMP$CHAR = INT$BUF;
366   2               TEMP$CHAR = TEMP$CHAR and OFH;
367   2               if TEMP$CHAR < 10
368   2                   then ASCII$BUF(O) = TEMP$CHAR + 30H;
369   2               else if (10 <= TEMP$CHAR) and (TEMP$CHAR <= OFH)
370   2                   then ASCII$BUF(O) = TEMP$CHAR + 37H;

371   2                   TEMP$CHAR = shr((INT$BUF and OFOH), 4);
372   2               if TEMP$CHAR < 10
373   2                   then ASCII$BUF(1) = TEMP$CHAR + 30H;
374   2                   else if (10 <= TEMP$CHAR) and (TEMP$CHAR <= OFH)
375   2                   then ASCII$BUF(1) = TEMP$CHAR + 37H;

376   2       end INT$TO$ASCII;


     /* ASCII code to integer value conversion

                        Name:       ASCII$TO$INT
                        Input:      ASCII$BUF    (2 bytes)
                        Output:     INT$BUF      (1 byte)
                        Function:   ASCII to INTEGER conversion */

377   1       ASCII$TO$INT: procedure;
378   2           declare TEMP$CHAR (2)             byte;

379   2                   TEMP$CHAR (O) = ASCII$BUF(O);
380   2                   if ('O' <= TEMP$CHAR(O)) and (TEMP$CHAR(O) <= '9')
381   2                       then TEMP$CHAR(O) = (TEMP$CHAR(O) - 30H);
382   2                   else if ('A' <= TEMP$CHAR(O)) and (TEMP$CHAR(O) <= 'F')
383   2                       then TEMP$CHAR(O) = TEMP$CHAR(O) - 37H;
384   2                   TEMP$CHAR (1) = ASCII$BUF(1);
385   2                   if ('O' <= TEMP$CHAR(1)) and (TEMP$CHAR(1) <= '9')
386   2                       then TEMP$CHAR(1) = TEMP$CHAR(1) - 30H;
387   2                   else if ('A' <= TEMP$CHAR(1)) and (TEMP$CHAR(1) <= 'F')
388   2                       then TEMP$CHAR(1) = TEMP$CHAR(1) - 37H;
389   2               INT$BUF = (TEMP$CHAR(O) and OFH) or shl((TEMP$CHAR(1) and OFH), 4);

390   2       end ASCII$TO$INT;


391   1       DEC$TO$HEX: procedure(NUMBER) word;

392   2           declare NUMBER       word;
393   2           declare TEMP$WORDO   word;
394   2           declare TEMP$WORD1   word;
```

296170-30

Table 6. Software Listings (Continued)

```
395   2               TEMP$WORD0 = shr((NUMBER and 0F000H),12);
396   2               TEMP$WORD1 = TEMP$WORD0 * 1000;
397   2               TEMP$WORD0 = shr((NUMBER and 0F00H),8);
398   2               TEMP$WORD1 = TEMP$WORD1 + (TEMP$WORD0 * 100);
399   2               TEMP$WORD0 = shr((NUMBER and 0F0H),4);
400   2               TEMP$WORD1 = TEMP$WORD1 + (TEMP$WORD0 * 10);
401   2               TEMP$WORD0 = NUMBER and 0FH;
402   2               TEMP$WORD1 = TEMP$WORD1 + TEMP$WORD0;
403   2               return TEMP$WORD1;
404   2       end DEC$TO$HEX;


405   1       INPUT$WORD: procedure word;

406   2           declare IN$WORD      word;
407   2           declare I            byte;

408   2           call INS;
409   2           I, IN$WORD = 0;
410   2           do while (I < 4 and C$BUF(I) <> CR and C$BUF(I) <> LF);
411   3               ASCII$BUF(0) = C$BUF(I);
412   3               ASCII$BUF(1) = 0;
413   3               call ASCII$TO$INT;
414   3               IN$WORD = shl(IN$WORD,4) or double(INT$BUF);
415   3               I = I + 1;
416   3           end;
417   2           return IN$WORD;
418   2       end INPUT$WORD;

    /* Read 1 byte data from Memory
                          Name:        MEM$BYTE$READ
                          Input:       CMD$OFFSET
                          Output:      MSG$BUF (console out)
                          Function:    Output one byte data stored in memory to
                                       console                              */


419   1       MEM$BYTE$READ: procedure (CMD$OFFSET);
420   2           declare
                          CMD$DAT based CMD$PTR     byte,
                          COUNT                     byte,
                          CMD$OFFSET                word;

421   2               CMD$PTR = build$ptr(COMMON$BASE,CMD$OFFSET);
422   2               DAT = CMD$DAT;
423   2               INT$BUF = CMD$DAT;
424   2               call INT$TO$ASCII;
425   2               call CO(ASCII$BUF(1));
426   2               call CO(ASCII$BUF(0));

427   2       and MEM$BYTE$READ;


    /* Receive buffer data display

                          Name:        BUFFER$DISP
                          Input:       C$BUF (console in)
                          Output:      MSG$BUF (console out)
                          Function:    Display Receive buffer data to console */


428   1       BUFFER$DISP: procedure;
429   2           declare BYTE$COUNT            byte;
430   2           declare TEMP$CMD$OFFSET       word;
431   2           declare I                     byte;
432   2           declare J                     byte;

433   2   LOOP7:  MSG$PTR = @(0DH,0AH,'RXB> KEY IN THE BUFFER DESCRIPTOR NUMBER (0 - F: 128 byte unit
                  )',0DH, 0AH,2AH,0);
434   2               call OUTS;
435   2   LOOP8:      call INS;
436   2               call CR$LF;
437   2               ASCII$BUF(0) = C$BUF(0);
438   2               ASCII$BUF(1) = 0;
439   2               call ASCII$TO$INT;
440   2               if INT$BUF <= 15 then do;
442   3                   TEMP$CMD$OFFSET = RX$OFFSET + (INT$BUF * 80h);
443   3                   do I = 0 to 7;
444   4                       INT$BUF = I;
445   4                       call INT$TO$ASCII;
446   4                       call CO(C$BUF(0));
447   4                       call CO(':');
```

296170-31

Table 6. Software Listings (Continued)

```
448   4                   call CO(ASCII$BUF(0));
449   4                   call CO('0');
450   4                   call CO(SP);
451   4                   do J = 0 to 15;
452   5                       call MEM$BYTE$READ(TEMP$CMD$OFFSET);
453   5                       call CO(SP);
454   5                       TEMP$CMD$OFFSET = TEMP$CMD$OFFSET + 1;
455   5                   end;
456   4                   call CR$LF;
457   4               end;
458   3           end;
459   2           else do;
460   3           call MSG$ILL$CMD;
461   3           MSG$PTR = @(0DH,0AH,'RXB> KEY IN 0 --- F(H) !!!',0DH,0AH,2AH,0);
462   3           goto LOOP8;
463   3       end;
464   2       MSG$PTR = @(0DH,0AH,'RXB> MORE RXB ? (Y or N)',0DH,0AH,2AH,0);
465   2       call OUTS;
466   2       if IN$YES then goto LOOP7;
468   2   end BUFFER$DISP;


    /* DUMP STATUS  status display

                            Name:       DS$BUFF$DISP
                            Input:      C$BUF (console in)
                            Output:     MSG$BUF (console out)
                            Function:   Display DUMP STATUS contents to Console */

469   1   DS$BUFF$DISP: procedure;
470   2       declare TEMP$CMD$OFFSET           word,
                      I                         byte,
                      J                         byte;

471   2       call CR$LF;
472   2       TEMP$CMD$OFFSET = DS$BUFF$OFFSET;
473   2       do I = 0 to 10;
474   3           INT$BUF = I;
475   3           call INT$TO$ASCII;
476   3           call CO(ASCII$BUF(1));
477   3           call CO(ASCII$BUF(0));
478   3           call CO('0');
479   3           call CO(SP);
480   3           do J = 0 to 15;
481   4               call MEM$BYTE$READ(TEMP$CMD$OFFSET);
482   4               call CO(SP);
483   4               TEMP$CMD$OFFSET = TEMP$CMD$OFFSET + 1;
484   4           end;
485   3           call CR$LF;
486   3       end;
487   2   end DS$BUFF$DISP;

    /* Transmit data change

                            Name:       TRANSMIT$DATA$CHANGE
                            Input:      C$BUF (console in)
                            Output:     Transmit buffer
                            Function:   Determine the transmit buffer size
                                        (128 bytes unit) and data */

488   1   TRANSMIT$DATA$CHANGE: procedure;
489   2       declare I                 byte;
490   2       declare BYTE$DATA         byte;
491   2       declare BUFFER$NUMBER     byte;
492   2       declare BUFFER$POINTER    pointer;
493   2       declare C$COUNT           word;
494   2       declare TEMP              word;

495   2   LOOP:   MSG$PTR = @(0DH,0AH,'TXB> DO YOU WANT TO CHANGE THE TRANSMIT DATA ?
                  (Y or N)',0DH,0AH,2AH,0);
496   2           call OUTS;
497   2           if IN$YES then
498   2           do;
499   3               MSG$PTR = @(0DH,0AH,'TXB> DATA CHANGE: KEY IN THE BUFFER NUMBER
                          (0 - F: 128 bytes unit)', 0DH,0AH,2AH,0);
500   3               call OUTS;
501   3               INT$BUF = LOW(INPUT$WORD);
502   3               BUFFER$NUMBER = INT$BUF and 0FH;
503   3               BUFFER$POINTER = @TX(BUFFER$NUMBER);
504   3               MSG$PTR = @(0DH,0AH,'TXB> KEY IN THE VALUE (00 - FF)', 0DH,0AH,2AH,0)
505   3               call OUTS;
506   3               INT$BUF = low(INPUT$WORD);
507   3               call setb(INT$BUF,BUFFER$POINTER,80H);
508   3               MSG$PTR = @(0DH,0AH,'TXB> DATA WAS WRITTEN',0DH,0AH,0);
509   3               call OUTS;
```

296170-32

Table 6. Software Listings (Continued)

```
510   3                    call CR$LF;
511   3                    goto LOOP;
512   3                end;
513   2                else do;
514   3                    MSG$PTR = @(0DH,0AH,'TXB> HOW MANY BYTE(S) DO YOU NEED ?',
                             0);
515   3                    call OUTS;
516   3                    MSG$PTR = @(0DH,0AH,'      KEY IN THE DECIMAL NUMBER'
                                   ,0DH,0AH,2AH,0);
517   3                    call OUTS;
518   3                    TEMP = INPUT$WORD;
519   3                    C$COUNT = DEC$TO$HEX(TEMP);
520   3                    I = 0;
521   3                    do while (C$COUNT > 80H and C$COUNT <= 2048);
522   4                        C$COUNT = C$COUNT - 80H;
523   4                        TBD(I).ACT$COUNT = 80H;
524   4                        I = I + 1;
525   4                    end;
526   3                    if C$COUNT <= 80H
527   3                        then TBD(I).ACT$COUNT = 8000H + C$COUNT;
528   3                    else if C$COUNT > 2048 then
529   3                    do;
530   4                        call CR$LF;
531   4                        MSG$PTR = @('TXB> BYTE COUNT IS LIMITED UP TO 2048',0DH,
                                     0AH,0);
532   4                        call OUTS;
533   4                    end;
534   3                end;

535   2        end TRANSMIT$DATA$CHANGE;

      /* Display command word contents

                          Name:       DISP$CONTENTS
                          Input:      C$BUF (console in)
                          Output:     MSG$BUF (console out)
                          Function:   Display the command word contents to console */

536   1        DISP$CONTENTS: procedure;

537   2            declare DISP$OFFSET          word;
538   2            declare BYTE$COUNT           byte;
539   2            declare J                    byte;

540   2            MSG$PTR = @(0DH,0AH,'CSET> DO YOU WANT TO SEE THE CONTENTS ? (Y or N)',0DH,
                       0AH,2AH,0);
541   2            call OUTS;
542   2            DISP$OFFSET = MESSAGE$WORD;

543   2            if IN$YES then
544   2            do;
545   3                J = (MESSAGE$BYTE - 2)/2;
546   3                do BYTE$COUNT = 0 to J;
547   4                    call CR$LF;
548   4                    if MESSAGE$STATUS = TRUE then
549   4                    do;
550   5                        MESSAGE$STATUS = FALSE;
551   5                        MSG$PTR = @('WORD ',0);
552   5                        call OUTS;
553   5                        INT$BUF = BYTE$COUNT;
554   5                        call INT$TO$ASCII;
555   5                        call CO(ASCII$BUF(0));
556   5                        call CO(SP);
557   5                        call CO('=');
558   5                        call CO(SP);
559   5                        DISP$OFFSET = MESSAGE$WORD + 1;
560   5                        call MEM$BYTE$READ(DISP$OFFSET);
561   5                        DISP$OFFSET = DISP$OFFSET - 1;
562   5                        call MEM$BYTE$READ(DISP$OFFSET);
563   5                        call CO('H');
564   5                    end;
565   4                    else do;
566   5                        MSG$PTR = @('WORD ',0);
567   5                        call OUTS;
568   5                        INT$BUF = BYTE$COUNT;
569   5                        call INT$TO$ASCII;
570   5                        call CO(ASCII$BUF(0));
571   5                        call CO(SP);
572   5                        call CO('=');
573   5                        call CO(SP);
574   5                        DISP$OFFSET = DISP$OFFSET + 3;
575   5                        call MEM$BYTE$READ(DISP$OFFSET);
576   5                        DISP$OFFSET = DISP$OFFSET - 1;
577   5                        call MEM$BYTE$READ(DISP$OFFSET);
578   5                        call CO('H');
579   5                    end;
```

296170-33

Table 6. Software Listings (Continued)

```
                  end;
              end;
          end DISP$CONTENTS;

      /* SCB STATUS, ACK field display

                              Name:       SCB$BITMAP$DISPLAY
                              Input:      SCB.STAT,SCB.CMD
                              Output:     MSG$BUF (console out)
                              Function:   Display the SCB STATUS, ACK field
                                          to console as binaly value */

          SCB$BITMAP$DISPLAY: procedure;
              declare I                        byte;

              do I = 0 to 3;
                  if rol(MESSAGE$WORD,I + 1)  then call CO(31H);
                  else call CO(30H);
              end;

          end SCB$BITMAP$DISPLAY;


      /* SCB COMMAND/STATUS field display

                              Name:       SCB$UNIT$DISPLAY
                              Input:      SCB.STAT,SCB.CMD
                              Output:     MSG$BUF (console out)
                              Function:   Display the CUS,RUS,CUC and RUC field
                                          to console as BCD value */

          SCB$UNIT$DISPLAY: procedure;
              declare SHIFT$COUNT              byte;

              if MESSAGE$STATUS = 0 then SHIFT$COUNT = 8;
              else SHIFT$COUNT = 4;
              INT$BUF = low(shr(MESSAGE$WORD,SHIFT$COUNT) and 7);
              call INT$TO$ASCII;
              call CO(ASCII$BUF(1));
              call CO(ASCII$BUF(0));

          end SCB$UNIT$DISPLAY;


      /* Word contents display

                              Name:       WORD$DISP
                              Input:      MESSAGE$WORD
                              Output:     MSG$BUF (console out)
                              Function:   display the word value */


          WORD$DISP:  procedure;
              declare I              byte;

                  INT$BUF = high(MESSAGE$WORD);
                  call INT$TO$ASCII;
                  call CO(ASCII$BUF(1));
                  call CO(ASCII$BUF(0));
                  INT$BUF = low(MESSAGE$WORD);
                  call INT$TO$ASCII;
                  call CO(ASCII$BUF(1));
                  call CO(ASCII$BUF(0));

          end WORD$DISP;


      /*change the link command condition

                              Name:       CHANGE
                              Input:      C$BUF (console in)
                              Output:     Command blocks
                              Function:   Change the command linking and command block
                                          parameter change                    */

          CHANGE: procedure;

              declare CHANGE$DATA$BYTE         word;
              declare CHANGE$PTR               pointer;
              declare CHANGE$OFFSET            word;
              declare BYTE$COUNT               byte;
              declare J                        byte;
              declare TMP$WORD                 word;
```

296170–34

Table 6. Software Listings (Continued)

```
619   2          TMP$WORD = MESSAGE$WORD;
620   2          call CR$LF;
621   2          MSG$PTR = @('CSET> DO YOU NEED CHANGE THIS BLOCK ? (Y or N)',ODH,OAH,2AH,O);
622   2          call OUTS;
623   2          if IN$YES then
624   2              do;
625   3                  CHANGE$OFFSET = MESSAGE$WORD;
626   3                  MESSAGE$STATUS = FALSE;
627   3                  do BYTE$COUNT = O to (MESSAGE$BYTE - 2) by 2;
628   4                      CHANGE$PTR = build$ptr(COMMON$BASE,CHANGE$OFFSET);
629   4                      call CR$LF;
630   4                      MSG$PTR = @('WORD ',O);
631   4                      call OUTS;
632   4                      INT$BUF = BYTE$COUNT / 2;
633   4                      call INT$TO$ASCII;
634   4                      call CO(ASCII$BUF(O));
635   4                      call CO(SP);
636   4                      call CO('=');
637   4                      call CO(SP);
638   4                      call movw(CHANGE$PTR,@MESSAGE$WORD,1);
639   4                      call WORD$DISP;
640   4                      MSG$PTR = @(20H,'<--- DO YOU NEED CHANGE THIS WORD?
                                      (Y or N)',ODH,OAH,2AH,O);
641   4                      call OUTS;
642   4                      if IN$YES then
643   4                      do;
644   5                          MSG$PTR = @(ODH,OAH,'CSET> WHAT IS THE VALUE ?  (FOUR DIGIT)',
                                      ODH,OAH,2AH,O);
645   5                          call OUTS;
646   5                          CHANGE$DATA$BYTE = INPUT$WORD;
647   5                          call setw(CHANGE$DATA$BYTE,CHANGE$PTR,1);
648   5                      end;
649   4                      CHANGE$OFFSET = CHANGE$OFFSET + 2;
650   4                  end;
651   3              end;
652   2              MESSAGE$WORD = TMP$WORD;
653   2          end CHANGE;


        /* Command set

                               Name:       LINK$ADDRESS$SET
                               Input:      CURRENT$CMD$OFFSET, NEXT$CMD$OFFSET
                               Output:     CURRENT$CMD$OFFSET, NEXT$CMD$OFFSET
                               Function:   Set the command link (Max. 2 commands)
                                           this routine sets the following parameter;
                                               a: link address set
                                               b: EL bit of previous command is reset
                                               c: EL bit of next command is set */

654   1          LINK$ADDRESS$SET: procedure;

655   2              declare TEMP$PTR                  pointer;
656   2              declare TEMP$WORD based TEMP$PTR  word;
657   2              declare TEMP$OFFSET               word;

658   2              if CMD$TOP$STATUS = FALSE then do;
660   3                  CMD$TOP$OFFSET = NEXT$CMD$OFFSET;
661   3                  CMD$TOP$STATUS = TRUE;
662   3              end;
663   2              else do;

              /* link address set */

664   3              TEMP$OFFSET = CURRENT$CMD$OFFSET + 4;
665   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
666   3              TEMP$WORD = NEXT$CMD$OFFSET;


              /* current$command EL-bit clear */

667   3              TEMP$OFFSET = CURRENT$CMD$OFFSET +2;
668   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
669   3              TEMP$WORD = TEMP$WORD and 7fffh;

              /* next command EL-bit set */

670   3              TEMP$OFFSET = MESSAGE$WORD  + 2;
671   3              TEMP$PTR = build$ptr(COMMON$BASE, TEMP$OFFSET);
672   3              TEMP$WORD = TEMP$WORD or 8000h;
673   3              end;
674   2          end LINK$ADDRESS$SET;
```

296170-35

Table 6. Software Listings (Continued)

```
               /* Transmit Command Link */

675   1          TRANSMIT$COMMAND$LINK: procedure;

676   2              declare I                                    word;

677   2              CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
678   2              NEXT$CMD$OFFSET = MESSAGE$WORD;
679   2              call LINK$ADDRESS$SET;
680   2              call CHANGE;
681   2              MESSAGE$STATUS = TRUE;
682   2              call DISP$CONTENTS;
683   2              I = O;
684   2      LOOP6:  if I <= 9 then do;
686   3                  MSG$PTR  = @(ODH,OAH,'CSET> MORE TRANSMIT COMMAND ? (Y or N)',
                                     ODH,OAH,2AH,O);
687   3                  call OUTS;
688   3                  if IN$YES then do;
690   4                      MESSAGE$WORD = OF500H + 16 * I;
691   4                      CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
692   4                      NEXT$CMD$OFFSET = MESSAGE$WORD;
693   4                      call LINK$ADDRESS$SET;
694   4                      call CHANGE;
695   4                      MESSAGE$STATUS = TRUE;
696   4                      call DISP$CONTENTS;
697   4                      I = I + 1;
698   4                      goto LOOP6;
699   4                  end;
700   3                  else do;
701   4                      call CR$LF;
702   4                      INT$BUF = I + 1;
703   4                      call INT$TO$ASCII;
704   4                      call CO(ASCII$BUF(O));
705   4                      MSG$PTR = @(' TRANSMIT COMMAND WERE SET',ODH,OAH,O);
706   4                      call OUTS;
707   4                  end;
708   3              end;
709   2              else do;
710   3                  EX$TRANSMIT(9).CMD = OA004H;
711   3                  EX$TRANSMIT(9).LINK$ADDRESS = OFFFFH;
712   3                  EX$TRANSMIT(9).BD$PTR = OF100H;
713   3                  MSG$PTR = @(ODH,OAH
                                     ,'TRANSMIT COMMAND LINKS ARE LIMITED UP TO 10 !!!'
                                     ,ODH,OAH,O);
714   3                  call OUTS;
715   3              end;
716   2          end TRANSMIT$COMMAND$LINK;


               /* Command link check

                                        Name:        COMMAND$LINK$CHECK
                                        Input:       MESSAGE$WORD
                                        Output:      LINK$ADDR
                                        Function:    Check the command link and pass the command
                                                     link condition to command set routine */

717   1          COMMAND$LINK$CHECK: procedure;
718   2              declare TEMP$CMD$OFFSET       word,
                             TEMP$CMD$WORD         word;

719   2              TEMP$CMD$WORD = MESSAGE$WORD;

720   2              if NEXT$CMD$OFFSET = MESSAGE$WORD then
721   2              do;
722   3                  MSG$PTR = @(ODH,OAH,'CSET> DO YOU WANT TO EXECUTE THE SAME COMMAND ? (Y or N)',
                             ODH,OAH,2AH,O);
723   3                  call OUTS;
724   3                  if IN$YES then
725   3                  do;
726   4                      call LINK$ADDRESS$SET;
727   4                      CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
728   4                      NEXT$CMD$OFFSET = MESSAGE$WORD;
729   4                  end;
730   3                  else do;
731   4                      MSG$PTR = @(ODH,OAH,'CSET> COMMAND LINK INPUT WAS CANCELLED',
                                 ODH,OAH,O);
732   4                      call OUTS;
733   4                  end;
734   3              end;
735   2              else do;
736   3                  CURRENT$CMD$OFFSET = NEXT$CMD$OFFSET;
737   3                  NEXT$CMD$OFFSET = MESSAGE$WORD;
738   3                  call LINK$ADDRESS$SET;
739   3              end;
740   2              MESSAGE$WORD = TEMP$CMD$WORD;
```

296170-36

Table 6. Software Listings (Continued)

```
        end COMMAND$LINK$CHECK;


        MORE$COMMAND: procedure;

            MSG$PTR = @(ODH, OAH, 'CSET> DO YOU WANT MORE COMMAND ?
                         (Y or N)', ODH, OAH, 2AH, O);
            call OUTS;
        end MORE$COMMAND;


    /* Command set

                        Name:       COMMAND$SET
                        Input:      C$BUF (console in)
                        Output:     MSG$WORD, MSG$BYTE and MSG$STATUS
                        Function:   Interpret the Command set input from
                                    console and pass to command parameter set
                                    routine                                    */

        COMMAND$SET: procedure;

            declare BYTE$COUNT          byte;
            declare J                   byte;
        LOOP1:  MSG$PTR = @(ODH, OAH, 'CSET> WHAT IS THE COMMAND?', ODH, OAH, 'KEY IN H FOR HELP',
                ODH, OAH, 2AH, O));
            call OUTS;
            call INS;
            if (cmpb (@C$BUF(O), @('NOP', ODH), 4) = TRUEW)
                then do;
                    MESSAGE$STATUS = TRUE;
                    MESSAGE$WORD = NOP$OFFSET;
                    MESSAGE$BYTE = 6;
                    call NOP$CMD;
                    call COMMAND$LINK$CHECK;
                    call DISP$CONTENTS;
                    call MORE$COMMAND;
                    if IN$YES then goto LOOP1;
                end;


            else do;
                if (cmpb (@C$BUF(O), @('IA', ODH), 3) = TRUEW)
                then do;
                    MESSAGE$STATUS = TRUE;
                    MESSAGE$WORD = IA$SETUP$OFFSET;
                    MESSAGE$BYTE = 12;
                    call IA$SETUP$CMD;
                    call COMMAND$LINK$CHECK;
                    call CHANGE;
                    MESSAGE$STATUS = TRUE;
                    call DISP$CONTENTS;
                    call MORE$COMMAND;
                    if IN$YES then goto LOOP1;
                end;

            else do;
                if (cmpb (@C$BUF(O), @('CONF', ODH), 5) = TRUEW)
                then do;
                    MESSAGE$STATUS = TRUE;
                    MESSAGE$WORD = CONF$OFFSET;
                    MESSAGE$BYTE = 18;
                    call CONF$CMD;
                    call COMMAND$LINK$CHECK;
                    call CHANGE;
                    MESSAGE$STATUS = TRUE;
                    call DISP$CONTENTS;
                    call MORE$COMMAND;
                    if IN$YES then goto LOOP1;
                end;
            else do;
                if (cmpb (@C$BUF(O), @('MC', ODH), 3) = TRUEW)
                then do;
                    MESSAGE$STATUS = TRUE;
                    MESSAGE$WORD = MC$SETUP$OFFSET;
                    MESSAGE$BYTE = 14;
                    call MC$SETUP$CMD;
                    call COMMAND$LINK$CHECK;
                    call CHANGE;
                    MESSAGE$STATUS = TRUE;
                    call DISP$CONTENTS;
                    call MORE$COMMAND;
                    if IN$YES then goto LOOP1;
                end;
```

296170-37

Table 6. Software Listings (Continued)

```
809   5            else do;
810   6                 if (cmpb (@C$BUF(0), @('TX',ODH), 3) = TRUEW)
811   6                 then do;
812   7                      MESSAGE$STATUS = TRUE;
813   7                      MESSAGE$WORD = TRANSMIT$OFFSET;
814   7                      MESSAGE$BYTE = 16;
815   7                      call TX$CMD;
816   7                      call EX$TX$CMD;
817   7                      call TRANSMIT$COMMAND$LINK;
818   7                      call MORE$COMMAND;
819   7                      if IN$YES then goto LOOP1;
821   7                 end;

822   6            else do;
823   7                 if (cmpb (@C$BUF(0), @('TDR',ODH), 4) = TRUEW)
824   7                 then do;
825   8                      MESSAGE$STATUS = TRUE;
826   8                      MESSAGE$WORD = TDR$OFFSET;
827   8                      MESSAGE$BYTE = 8;
828   8                      call TDR$CMD;
829   8                      call COMMAND$LINK$CHECK;
830   8                      call CHANGE;
831   8                      MESSAGE$STATUS = TRUE;
832   8                      call DISP$CONTENTS;
833   8                      call MORE$COMMAND;
834   8                      if IN$YES then goto LOOP1;
836   8                 end,

837   7            else do;
838   8                 if (cmpb (@C$BUF(0), @('DS',ODH), 3) = TRUEW)
839   8                 then do;
840   9                      MESSAGE$STATUS = TRUE;
841   9                      MESSAGE$WORD = DUMP$STAT$OFFSET;
842   9                      MESSAGE$BYTE = 8;
843   9                      call DUMP$STAT$CMD;
844   9                      call COMMAND$LINK$CHECK;
845   9                      call CHANGE;
846   9                      MESSAGE$STATUS = TRUE;
847   9                      call DISP$CONTENTS;
848   9                      call MORE$COMMAND;
849   9                      if IN$YES then goto LOOP1;
851   9                 end;

852   8            else do;
853   9                 if (cmpb (@C$BUF(0), @('DIAG',ODH), 5) = TRUEW)
854   9                 then do;
855  10                      MESSAGE$STATUS = TRUE;
856  10                      MESSAGE$WORD = DIAG$OFFSET;
857  10                      MESSAGE$BYTE = 6;
858  10                      call DIAG$CMD;
859  10                      call COMMAND$LINK$CHECK;
860  10                      call CHANGE;
861  10                      MESSAGE$STATUS = TRUE;
862  10                      call DISP$CONTENTS;
863  10                      call MORE$COMMAND;
864  10                      if IN$YES then goto LOOP1;
866  10                 end;

867   9            else do;
868  10                 if (cmpb(@C$BUF(0),@('H',ODH), 2) = TRUEW)
869  10                      then do;
870  11                           call COMMAND$MSG;
871  11                           call CR$LF;
872  11                           goto LOOP1;
873  11                      end;
874  10            else do;
875  11                 if (cmpb(@C$BUF(0),@('EXIT',ODH),5) = TRUEW)
876  11                 then call EXIT;
877  11                 else      call MSG$ILL$CMD;
878  11                                                   end;
879  10                                              end;
880   9                                         end;
881   8                                    end;
882   7                               end;
883   6                          end;
884   5                     end;
885   4                end;
886   3           end;
887   2      end COMMAND$SET;
```

296170-38

Table 6. Software Listings (Continued)

```
                    /* SCP, ISCP contents display */

888    1                CP$DISPLAY: procedure;
889    2                    MSG$PTR = @('      *** SCP ***', ODH, OAH, O);
890    2                    call OUTS;
                    /* SCP */
891    2                    MESSAGE$WORD = SCP.SYSBUS;
892    2                    call WORD$DISP;
893    2                    MSG$PTR = @('      SCP SYSBUS', ODH, OAH, O);
894    2                    call OUTS;
895    2                    MESSAGE$WORD = SCP.ISCP$OFFSET;
896    2                    call WORD$DISP;
897    2                    MSG$PTR = @('      ISCP OFFSET', ODH, OAH, O);
898    2                    call OUTS;
899    2                    MESSAGE$WORD = SCP.ISCP$BASE;
900    2                    call WORD$DISP;
901    2                    MSG$PTR = @('      ISCP BASE', ODH, OAH, O);
902    2                    call OUTS;
                    /* ISCP */
903    2                    MSG$PTR = @('      *** ISCP ***', ODH, OAH, O);
904    2                    call OUTS;
905    2                    MESSAGE$WORD = ISCP.BUSY;
906    2                    call WORD$DISP;
907    2                    MSG$PTR = @('      ISCP BUSY', ODH, OAH, O);
908    2                    call OUTS;
909    2                    MESSAGE$WORD = ISCP.SCB$OFFSET;
910    2                    call WORD$DISP;
911    2                    MSG$PTR = @('      SCB OFFSET', ODH, OAH, O);
912    2                    call OUTS;
913    2                    MESSAGE$WORD = ISCP.SCB$BASE1;
914    2                    call WORD$DISP;
915    2                    MSG$PTR = @('      SCB BASE L', ODH, OAH, O);
916    2                    call OUTS;
917    2                    MESSAGE$WORD = ISCP.SCB$BASE2;
918    2                    call WORD$DISP;
919    2                    MSG$PTR = @('      SCB BASE H', ODH, OAH, O);
920    2                    call OUTS;

921    2                end CP$DISPLAY;


                    /* All of SCB information display

                                    Name:        SCB$STATUS$DISPLAY
                                    Input:       SCB
                                    Output:      MSG$BUF (console out)
                                    Function:    Pass contents of SCB information to
                                                 SCB contents display routine                   */

922    1                SCB$STATUS$DISPLAY: procedure;

923    2                    MSG$PTR = @('      *** SCB ***', ODH, OAH, O);
924    2                    call OUTS;
925    2                    MESSAGE$WORD = SCB.STAT;
926    2                    call SCB$BITMAP$DISPLAY;
927    2                    MSG$PTR = @('B     STATUS', ODH, OAH, O);
928    2                    call OUTS;
929    2                    MESSAGE$WORD = SCB.CMD;
930    2                    call SCB$BITMAP$DISPLAY;
931    2                    MSG$PTR = @('B     ACK', ODH, OAH, O);
932    2                    call OUTS;
933    2                    MESSAGE$WORD = SCB.STAT;
934    2                    MESSAGE$STATUS = 0;
935    2                    call SCB$UNIT$DISPLAY;
936    2                    MSG$PTR = @('H         CUS', ODH, OAH, O);
937    2                    call OUTS;
938    2                    MESSAGE$WORD = SCB.STAT;
939    2                    MESSAGE$STATUS = 1;
940    2                    call SCB$UNIT$DISPLAY;
941    2                    MSG$PTR = @('H         RUS', ODH, OAH, O);
942    2                    call OUTS;
943    2                    MESSAGE$WORD = SCB.CMD;
944    2                    MESSAGE$STATUS = 0;
945    2                    call SCB$UNIT$DISPLAY;
946    2                    MSG$PTR = @('H         CUC', ODH, OAH, O);
947    2                    call OUTS;
948    2                    MESSAGE$WORD = SCB.CMD;
949    2                    MESSAGE$STATUS = 1;
950    2                    call SCB$UNIT$DISPLAY;
951    2                    MSG$PTR = @('H         RUC', ODH, OAH, O);
952    2                    call OUTS;
953    2                    MESSAGE$WORD = SCB.CBL$OFFSET;
954    2                    call WORD$DISP;
955    2                    MSG$PTR = @('H     CBL OFFSET', ODH, OAH, O);
956    2                    call OUTS;
```

296170-39

Table 6. Software Listings (Continued)

```
2               MESSAGE$WORD = SCB.RFD$OFFSET;
2               call WORD$DISP;
2               MSG$PTR = @('H  RFD OFFSET',ODH,OAH,0);
2               call OUTS;
2               MESSAGE$WORD = SCB.CRC$ERRS;
2               call WORD$DISP;
2               MSG$PTR = @('H  CRC ERRORS',ODH,OAH,0);
2               call OUTS;
2               MESSAGE$WORD = SCB.ALN$ERRS;
2               call WORD$DISP;
2               MSG$PTR = @('H  ALIGNMENT ERRORS',ODH,OAH,0);
2               call OUTS;
2               MESSAGE$WORD = SCB.RSC$ERRS;
2               call WORD$DISP;
2               MSG$PTR = @('H  NO RESORCE ERRORS',ODH,OAH,0);
2               call OUTS;
2               MESSAGE$WORD = SCB.OVRN$ERRS;
2               call WORD$DISP;
2               MSG$PTR = @('H  OVERRUN ERRORS',0);
2               call OUTS;
2           end SCB$STATUS$DISPLAY;

        /* Action commands status display

                           Name:       ACTION$CMD$STATUS$DISPLAY
                           Input:      Action commands status word
                           Output:     MESSAGE$WORD
                           Function:   Pass the action command status display
                                       parameter to WORD$DISPLAY routine.
                                       This command performs all the action command
                                       status display at once.                    */

1           ACTION$CMD$STATUS$DISPLAY: procedure;

2               declare     I    byte;

2               call CR$LF;
2               MSG$PTR = @('NOP ------------ ',0);
2               call OUTS;
2               MESSAGE$WORD = NOP.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('IA SETUP ------- ',0);
2               call OUTS;
2               MESSAGE$WORD = IA$SETUP.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('CONFIGURATION -- ',0);
2               call OUTS;
2               MESSAGE$WORD = CONF.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('MC SETUP ------- ',0);
2               call OUTS;
2               MESSAGE$WORD = MC$SETUP.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('TRANSMIT ------- ',0);
2               call OUTS;
2               MESSAGE$WORD = TRANSMIT.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('TDR ------------ ',0);
2               call OUTS;
2               MESSAGE$WORD = TDR.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('DUMP STATUS ---- ',0);
2               call OUTS;
2               MESSAGE$WORD = DUMP$STAT.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
2               MSG$PTR = @('DIAGNOSE ------- ',0);
2               call OUTS;
2               MESSAGE$WORD = DIAG.STAT;
2               call WORD$DISP;
2               call CO('H');
2               call CR$LF;
```

296170-40

Table 6. Software Listings (Continued)

```
1029   2          MSG$PTR = @('EXTRA TRANSMIT STATUS',ODH,OAH,ODH,OAH,0);
1030   2          call OUTS;
1031   2          do I = 0 to 9;
1032   3              MESSAGE$WORD = EX$TRANSMIT(I).STAT;
1033   3              call WORD$DISP;
1034   3              call CO('H');
1035   3              call CO(SP);
1036   3          end;
1037   2          call CR$LF;
1038   2      end ACTION$CMD$STATUS$DISPLAY;

          /* Time Domain Refrect-meter result display

                            Name:       TDR$RESULT
                            Input:      TDR.TIME
                            Output:     MESSAGE$WORD
                            Function:   Pass the TDR command execution result to
                                        WORD$DISP routine.
                                        This routine performs TDR result display */
1039   1      TDR$RESULT: procedure;

1040   2          MESSAGE$WORD = TDR.TIME;
1041   2          MSG$PTR = @(ODH,OAH,'TDR RESULT = ',0);
1042   2          call OUTS;
1043   2          call WORD$DISP;
1044   2          call CR$LF;
1045   2      end TDR$RESULT;

          /* Buffer status display

                            Name:       BD$STATUS$DISPLAY
                            Input:      All of buffer descriptors status and
                                        actual count field.
                            Output:     MESSAGE$WORD
                            Function:   Pass the parameters of all of buffer descriptors
                                        status and actual count to WORD$DISP routine. */


1046   1      BD$STATUS$DISPLAY: procedure;


1047   2          declare I                    byte;

1048   2          MSG$PTR = @(ODH,OAH,'** RFD STATUS',ODH,OAH,0);
1049   2          call OUTS;
1050   2          do I = 0 to 15;
1051   3              MESSAGE$WORD = RFD(I).STAT;
1052   3              call WORD$DISP;
1053   3              call CO(SP);
1054   3          end;

1055   2          MSG$PTR = @('** RFD EL AND S',ODH,OAH,0);
1056   2          call OUTS;
1057   2          do I = 0 to 15;
1058   3              MESSAGE$WORD = RFD(I).EL$S;
1059   3              call WORD$DISP;
1060   3              call CO(SP);
1061   3          end;

1062   2          MSG$PTR = @('** RFD LINK ADDRESS',ODH,OAH,0);
1063   2          call OUTS;
1064   2          do I = 0 to 15;
1065   3              MESSAGE$WORD = RFD(I).LINK$ADDRESS;
1066   3              call WORD$DISP;
1067   3              call CO(SP);
1068   3          end;

1069   2          MSG$PTR = @('** RFD BUFFER DESCRIPTOR POINTER',ODH,OAH,0);
1070   2          call OUTS;
1071   2          do I = 0 to 15;
1072   3              MESSAGE$WORD = RFD(I).BD$PTR;
1073   3              call WORD$DISP;
1074   3              call CO(SP);
1075   3          end;

1076   2          MSG$PTR = @('** RFD DESTINATION ADDRESS',ODH,OAH,0);
1077   2          call OUTS;
1078   2          do I = 0 to 15;
1079   3              MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(1)),8) +
                                    double(RFD(I).DEST$ADDRESS(0)));
1080   3              call WORD$DISP;
1081   3              call CO(SP);
1082   3          end;
```

296170-41

Table 6. Software Listings (Continued)

```
        do I = 0 to 15;
            MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(3)),8) +
                          double(RFD(I).DEST$ADDRESS(2)));
            call WORD$DISP;
            call CO(SP);
        end;
        do I = 0 TO 15;
            MESSAGE$WORD = (shl(double (RFD(I).DEST$ADDRESS(5)),8) +
                          double(RFD(I).DEST$ADDRESS(4)));
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** RFD SOURCE ADDRESS',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(1)),8) +
                          double(RFD(I).SOURCE$ADDRESS(0)));
            call WORD$DISP;
            call CO(SP);
        end;
        do I = 0 to 15;
            MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(3)),8) +
                          double(RFD(I).SOURCE$ADDRESS(2)));
            call WORD$DISP;
            call CO(SP);
        end;
        do I = 0 to 15;
            MESSAGE$WORD = (shl(double (RFD(I).SOURCE$ADDRESS(5)),8) +
                          double(RFD(I).SOURCE$ADDRESS(4)));
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** RFD TYPE FIELD',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = RFD(I).TYPE$FIELD;
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @(ODH,OAH,' TYPE <CR> TO CONTINUE',0);
        call OUTS;
        call INS;
        MSG$PTR = @(ODH,OAH,'** RBD ACTUAL COUNT',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = RBD(I).ACT$COUNT;
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** RBD NEXT BD ADDRESS',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = RBD(I).NEXT$BD$ADD;
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** RBD BUFFER OFFSET, BASE',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = RBD(I).BUFF$OFFSET;
            call WORD$DISP;
            call CO(SP);
        end;

        do I = 0 to 15;
            MESSAGE$WORD = RBD(I).BUFF$BASE;
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** RBD BUFFER SIZE',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
            MESSAGE$WORD = RBD(I).SIZE;
            call WORD$DISP;
            call CO(SP);
        end;

        MSG$PTR = @('** TBD ACTUAL COUNT',ODH,OAH,0);
        call OUTS;
        do I = 0 to 15;
```

296170-42

Table 6. Software Listings (Continued)

```
1156  3                  MESSAGE$WORD = TBD(I).ACT$COUNT;
1157  3                  call WORD$DISP;
1158  3                  call CO(SP);
1159  3              end;

1160  2              MSG$PTR = @('** TBD NEXT BD ADDRESS',ODH,OAH,0);
1161  2              call OUTS;
1162  2              do I = 0 to 15;
1163  3                  MESSAGE$WORD = TBD(I).NEXT$BD$ADD;
1164  3                  call WORD$DISP;
1165  3                  call CO(SP);
1166  3              end;

1167  2              MSG$PTR = @('** TBD BUFFER OFFSET,BASE',ODH,OAH,0);
1168  2              call OUTS;
1169  2              do I = 0 to 15;
1170  3                  MESSAGE$WORD = TBD(I).BUFF$OFFSET;
1171  3                  call WORD$DISP;
1172  3                  call CO(SP);
1173  3              end;
1174  2              do I = 0 to 15;
1175  3                  MESSAGE$WORD = TBD(I).BUFF$BASE;
1176  3                  call WORD$DISP;
1177  3                  call CO(SP);
1178  3              end;

1179  2          end BD$STATUS$DISPLAY;

1180  1          IN$STAT: procedure byte;
1181  2              declare CHAR            byte;
1182  2              do while ((CHAR <> 'A' or CHAR <> 'a') or
                              (CHAR <> 'B' or CHAR <> 'b') or
                              (CHAR <> 'S' or CHAR <> 's'));
1183  3              CHAR = CI and 7FH;
1184  3              call CO(CHAR);
1185  3              if CHAR = 'A' or CHAR = 'a'
1186  3                  then return 'A';
1187  3              else if CHAR = 'B' or CHAR = 'b'
1188  3                  then return 'B';
1189  3              else if CHAR = 'S' or CHAR = 's'
1190  3                  then return 'S';
1191  3              else do;
1192  4                  MSG$PTR = @(ODH,OAH,'STAT> KEY IN S, A, or B',ODH,OAH,2AH,0);
1193  4                  call OUTS;
1194  4                  end;
1195  3              end;
1196  2          end IN$STAT;


      /* Status display

                                Name:       STATUS$DISPLAY
                                Input:      C$BUF (console in)
                                Output:     None
                                Function:   Interpret the key input command what status
                                            should be displayed                          */

1197  1          STATUS$DISPLAY: procedure;
1198  2              declare I                 byte;

1199  2              MSG$PTR = @(ODH,OAH,'STAT> KEY IN COMMAND THAT YOU WANT TO SEE IN THE STATUS',
                         ODH,OAH,0);
1200  2              call OUTS;
1201  2              MSG$PTR = @('S FOR SCB, A FOR ACTION COMMANDS, B FOR BUFFER DESCRIPTORS',
                         ODH,OAH,2AH,0);
1202  2              call OUTS;
1203  2              MESSAGE$BYTE = IN$STAT;
1204  2              if MESSAGE$BYTE = 'A' then call ACTION$CMD$STATUS$DISPLAY;
1206  2              else if MESSAGE$BYTE = 'B' then call BD$STATUS$DISPLAY;
1208  2              else if MESSAGE$BYTE = 'S' then do;
1210  3                                              call CP$DISPLAY;
1211  3                                              call SCB$STATUS$DISPLAY;
1212  3              end;

1213  2          end STATUS$DISPLAY;

      /* Initialize individual block

                                Name:       INIT$COMMAND
                                Input:      C$BUF (console in)
                                Output:     None
```

296170-43

Table 6. Software Listings (Continued)

```
                              Function:         Interpret the key in command to determine
                                                what should be initialized individually.
                                                This routine is able to initialize
                                                the command block, buffer descriptors and
                                                buffers.                                        */
1214  1          INIT$COMMAND: procedure;

1215  2          LOOP3:  MSG$PTR = @(ODH,OAH,'INIT> KEY IN YOUR COMMAND (H FOR HELP)',
                         ODH,OAH,2AH,O);
1216  2              call OUTS;
1217  2              call INS;
1218  2              if (cmpb(@C$BUF(O), @('NOP',ODH), 4) = TRUEW) then call NOP$CMD;
1220  2              else if (cmpb(@C$BUF(O),@('IA',ODH), 3) = TRUEW) then call IA$SETUP$CMD;
1222  2              else if (cmpb(@C$BUF(O),@('CONF',ODH), 5) = TRUEW) then call CONF$CMD;
1224  2              else if (cmpb(@C$BUF(O),@('MC',ODH), 3) = TRUEW) then call MC$SETUP$CMD;
1226  2              else if (cmpb(@C$BUF(O),@('TX',ODH), 3) = TRUEW) then call TX$CMD;
1228  2              else if (cmpb(@C$BUF(O),@('TDR',ODH), 4) = TRUEW) then call TDR$CMD;
1230  2              else if (cmpb(@C$BUF(O),@('DS',ODH), 3) = TRUEW) then   call DUMP$STAT$CMD;
1232  2              else if (cmpb(@C$BUF(O),@('DIAG',ODH), 5) = TRUEW) then call DIAG$CMD;
1234  2              else if (cmpb(@C$BUF(O),@('RFD',ODH), 4) = TRUEW) then call INIT$RFD;
1236  2              else if (cmpb(@C$BUF(O),@('RBD',ODH), 4) = TRUEW) then call INIT$RBD;
1238  2              else if (cmpb(@C$BUF(O),@('RB',ODH), 3) = TRUEW)  then call INIT$RB;
1240  2              else if (cmpb(@C$BUF(O),@('TBD',ODH), 4) = TRUEW) then call INIT$TBD;
1242  2              else if (cmpb(@C$BUF(O),@('H',ODH), 1) = TRUEW) then
1243  3                  do; call COMMAND$MSG;
1245  3                      call BD$MSG;
1246  3                      call CR$LF;
1247  3                      goto LOOP3;
1248  3                  end;
1249  2              else    call MSG$ILL$CMD;

1250  2          end INIT$COMMAND;


1251  1          IN$CR: procedure byte;

1252  2              declare CHAR                        byte;
1253  2              CHAR = O;
1254  2              do while ((CHAR <> 'C') or (CHAR <> 'c') or
                              (CHAR <> 'R') or (CHAR <> 'r') or
                              (CHAR <> 'B') or (CHAR <> 'b'));
1255  3                  CHAR = CI and 7FH;
1256  3                  call CO(CHAR);
1257  3                  if CHAR = 'C' or CHAR = 'C'
1258  3                      then return 'C';
1259  3                  else if CHAR = 'R' or CHAR = 'r'
1260  3                      then return 'R';
1261  3                  else if CHAR = 'B' or CHAR = 'b'
1262  3                      then return 'B';
1263  3                  else do;
1264  4                      MSG$PTR = @(ODH,OAH,'** KEY IN C , R or B',ODH,OAH,2AH,O);
1265  4                      call OUTS;
1266  4                  end;
1267  3              end;
1268  2          end IN$CR;

         /* Unit control

                              Name:          UNIT$CONTROL
                              Input:         C$BUF (Console)
                              Output:        None
                              Function:      Controls the CU or RU execution.
                                             SCB information will be displayed.
                                             This command is used to control CU or RU      */
1269  1          UNIT$CONTROL: procedure(CONTROL);
1270  2              declare        CONTROL        byte;

1271  2              MSG$PTR = @(ODH,OAH,'KEY IN  C  FOR COMMAND,  R  FOR RECEIVE',
                                 ODH,OAH,2AH,O);
1272  2              call OUTS;
1273  2              MESSAGE$BYTE = IN$CR;
1274  2              if MESSAGE$BYTE = 'C' then
1275  2              do case CONTROL;
1276  3                  SCB.CMD = CMD$START;
1277  3                  SCB.CMD = CMD$ABORT;
1278  3                  SCB.CMD = CMD$SUSPEND;
1279  3                  SCB.CMD = CMD$RESUME;
1280  3              end;
1281  2              else if MESSAGE$BYTE = 'R' then
1282  2              do case CONTROL;
1283  3                  SCB.CMD = RCV$START;
1284  3                  SCB.CMD = RCV$ABORT;
1285  3                  SCB.CMD = RCV$SUSPEND;
```

296170-44

Table 6. Software Listings (Continued)

```
1286   3                 SCB.CMD = RCV$RESUME;
1287   3            end;
1288   2            if MESSAGE$BYTE = 'B' then
1289   2            do case CONTROL;
1290   3                 SCB.CMD = CMD$START or RCV$START;
1291   3                 SCB.CMD = CMD$ABORT or RCV$ABORT;
1292   3                 SCB.CMD = CMD$SUSPEND or RCV$SUSPEND;
1293   3                 SCB.CMD = CMD$RESUME or RCV$RESUME;
1294   3            end;
1295   2                 call CR$LF;
1296   2                 SCB.CBL$OFFSET = CMD$TOP$OFFSET;
1297   2                 CMD$TOP$STATUS = FALSE;
1298   2                 call SCB$STATUS$DISPLAY;
1299   2                 call CA;
1300   2       end UNIT$CONTROL;


        /* Master help display

                             Name:          MASTER$HELP
                             Input:         None
                             Output:        MSG$BUF (Console out)
                             Function:      This routine displays help for
                                             master command interpret routine.  */

1301   1       MASTER$HELP: procedure;
1302   2            call CR$LF;
1303   2            MSG$PTR = @('START THE UNIT(RU,CU) ---------- START
                           ABORT THE UNIT(RU,CU) ---------- ABORT',ODH,OAH,
                           'SUSPEND THE UNIT(RU,CU) ------ SUSPEND
                           RESUME THE UNIT(RU,CU) -------- RESUME',ODH,OAH,O);
1304   2            call OUTS;
1305   2            MSG$PTR = @('COMMAND BLOCK SET UP ------------ CSET
                           STATUS DISPLAY ------------------ STAT',ODH,OAH,
                           'INITIALIZE ---------------------- INIT
                           RECEIVE BUFFER DISPLAY ----------- RXB',ODH,OAH,O);
1306   2            call OUTS;
1307   2            MSG$PTR = @('TRANSMIT BUFFER DATA CHANGE ------ TXB',ODH,OAH,
                           'EXIT FROM THIS MONITOR ---------- EXIT
                           TDR RESULT ---------------------- TRES',ODH,OAH,
                           'DS RESULT BUFFER DISPLAY ------- DSRES  ',O);
1308   2            call OUTS;
1309   2            MSG$PTR = @('CHIP RESET ---------------------- CHRES',ODH,OAH,
                           'ACKNOWRIGE TO INTERRUPT ---------- ACK  ',O);
1310   2            call OUTS;
1311   2       end MASTER$HELP;


        /* Master command interpreter

                        Name:        MASTER$CMD
                        Input:       C$BUF (console in)
                        Output:      None
                        Function:    Interpret the First Level commands.  */

1312   1       MASTER$CMD: procedure;

1313   2       LOOP4:  MSG$PTR = @(ODH,OAH,'MAIN> KEY IN YOUR COMMAND (H FOR HELP)', ODH,OAH,2AH,O);
1314   2            call OUTS;
1315   2            call INS;

1316   2            if (cmpb (@C$BUF(O),@('START',ODH), 6) = TRUEW)
1317   2                 then call UNIT$CONTROL(O);
1318   2            else if (cmpb(@C$BUF(O),@('ABORT',ODH), 6) = TRUEW)
1319   2                 then call UNIT$CONTROL(1);
1320   2            else if (cmpb(@C$BUF(O),@('SUSPEND',ODH), 8) = TRUEW)
1321   2                 then call UNIT$CONTROL(2);
1322   2            else if (cmpb(@C$BUF(O),@('RESUME',ODH), 7) = TRUEW)
1323   2                 then call UNIT$CONTROL(3);
1324   2            else if (cmpb(@C$BUF(O),@('CSET',ODH), 5) = TRUEW)
1325   2                 then call COMMAND$SET;
1326   2            else if (cmpb(@C$BUF(O),@('STAT',ODH), 5 ) = TRUEW)
1327   2                 then call STATUS$DISPLAY;
1328   2            else if (cmpb(@C$BUF(O),@('INIT',ODH), 5) = TRUEW)
1329   2                 then call INIT$COMMAND;
1330   2            else if (cmpb(@C$BUF(O),@('RXB',ODH), 4) = TRUEW)
1331   2                 then call BUFFER$DISP;
1332   2            else if (cmpb(@C$BUF(O),@('TXB',ODH), 4) = TRUEW)
1333   2                 then call TRANSMIT$DATA$CHANGE;
1334   2            else if (cmpb(@C$BUF(O),@('H',ODH), 2) = TRUEW)
1335   2                 then do;
1336   3                      call MASTER$HELP;
1337   3                      goto LOOP4;
```

296170-45

Table 6. Software Listings (Continued)

```
                    end;
                 else if (cmpb(@C$BUF(0),@('EXIT',0DH), 5) = TRUEW)
                    then call EXIT;
                 else if (cmpb(@C$BUF(0),@('TRES',0DH), 5) = TRUEW)
                    then call TDR$RESULT;
                 else if (cmpb(@C$BUF(0),@('DSRES',0DH), 5) = TRUEW)
                    then call DS$BUFF$DISP;
                 else if (cmpb(@C$BUF(0),@('CHRES',0DH), 6) = TRUEW)
                    then call CHIP$RESET;
                 else if (cmpb(@C$BUF(0),@('ACK',0DH), 4) = TRUEW)
                    then call ACK;

                 else call MSG$ILL$CMD;
              end MASTER$CMD;


      /* Main Routine */


              MAIN: do;

                 call INIT$SYS;
                 call INIT$RFD;
                 call INIT$RBD;
                 call INIT$RB;
                 call INIT$TBD;
                 call INIT$TB;
                 CMD$TOP$STATUS = FALSE;
                 NEXT$CMD$OFFSET = TRUEW;

                 MSG$PTR = @('82586 DUAL PORT MEMORY TEST PROGRAM X206',0DH,0AH,0);
                 call OUTS;
                 do FOREVER;
                    call MASTER$CMD;
                 end;
              end main;
           end;


MODULE INFORMATION:

     CODE AREA SIZE     = 29C0H  10688D
     CONSTANT AREA SIZE = 0DF5H   3573D
     VARIABLE AREA SIZE = 0160H    352D
     MAXIMUM STACK SIZE = 0020H     32D
     2259 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:
     159KB MEMORY
```

296170–46

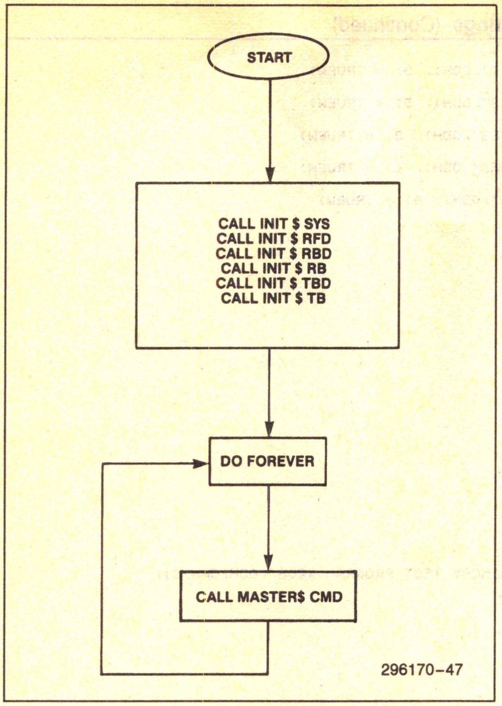


Figure 12A. Main Program Flow

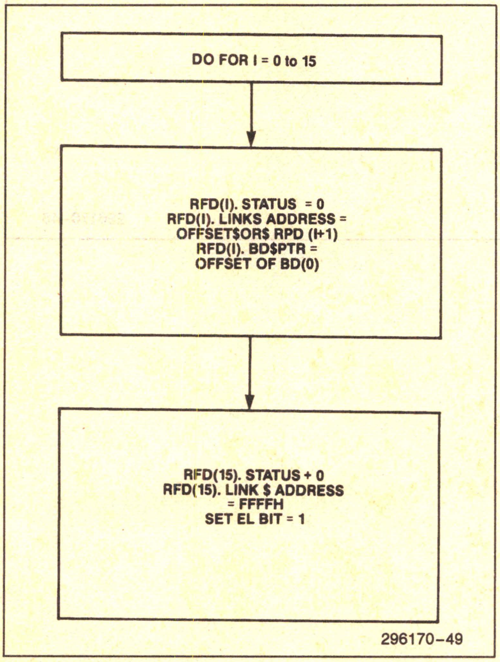


Figure 12C. Receive Frame Descriptor Initialization

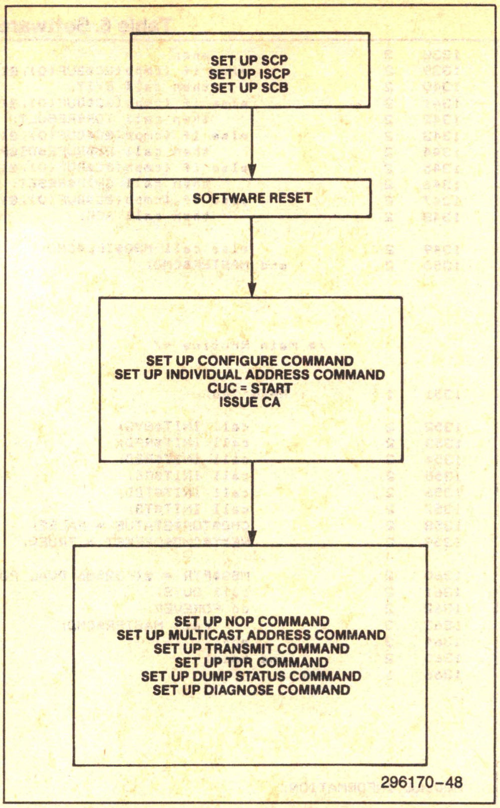


Figure 12B. System Initialization Routine

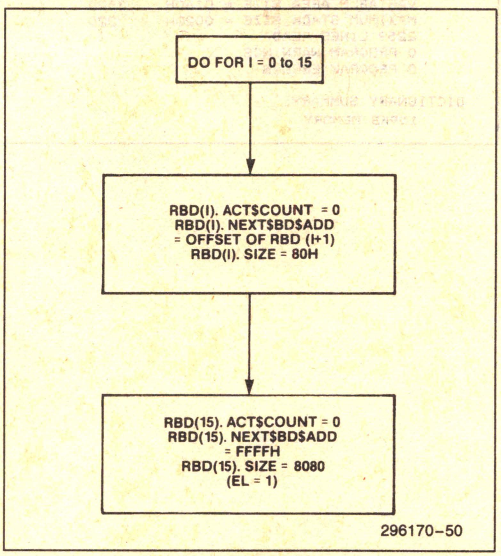


Figure 12D. Receive Buffer Descriptor Initialization

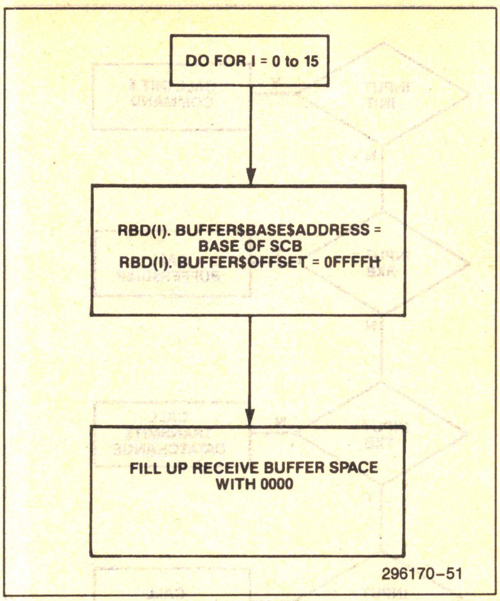


Figure 12E. Receive Buffer Space Initialization

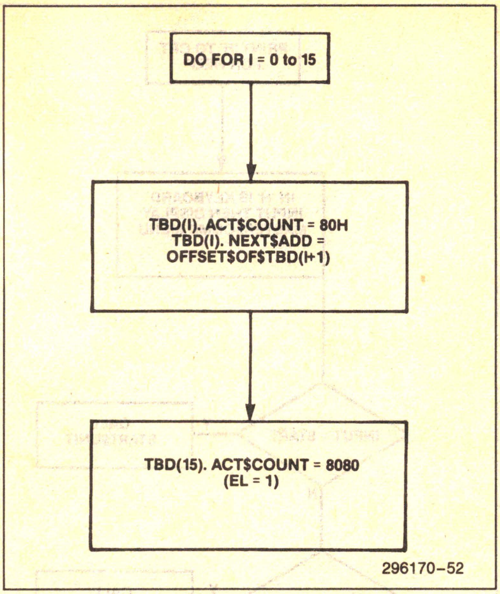


Figure 12F. Initialization of Transmit Buffer Descriptors

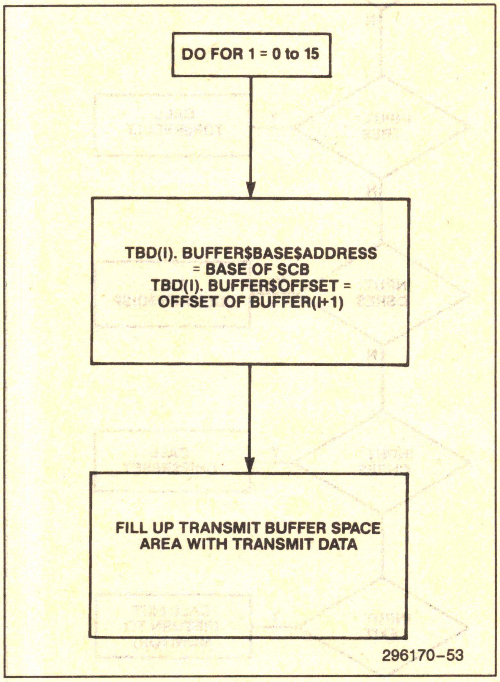


Figure 12G. Transmit Buffer Data Initialization

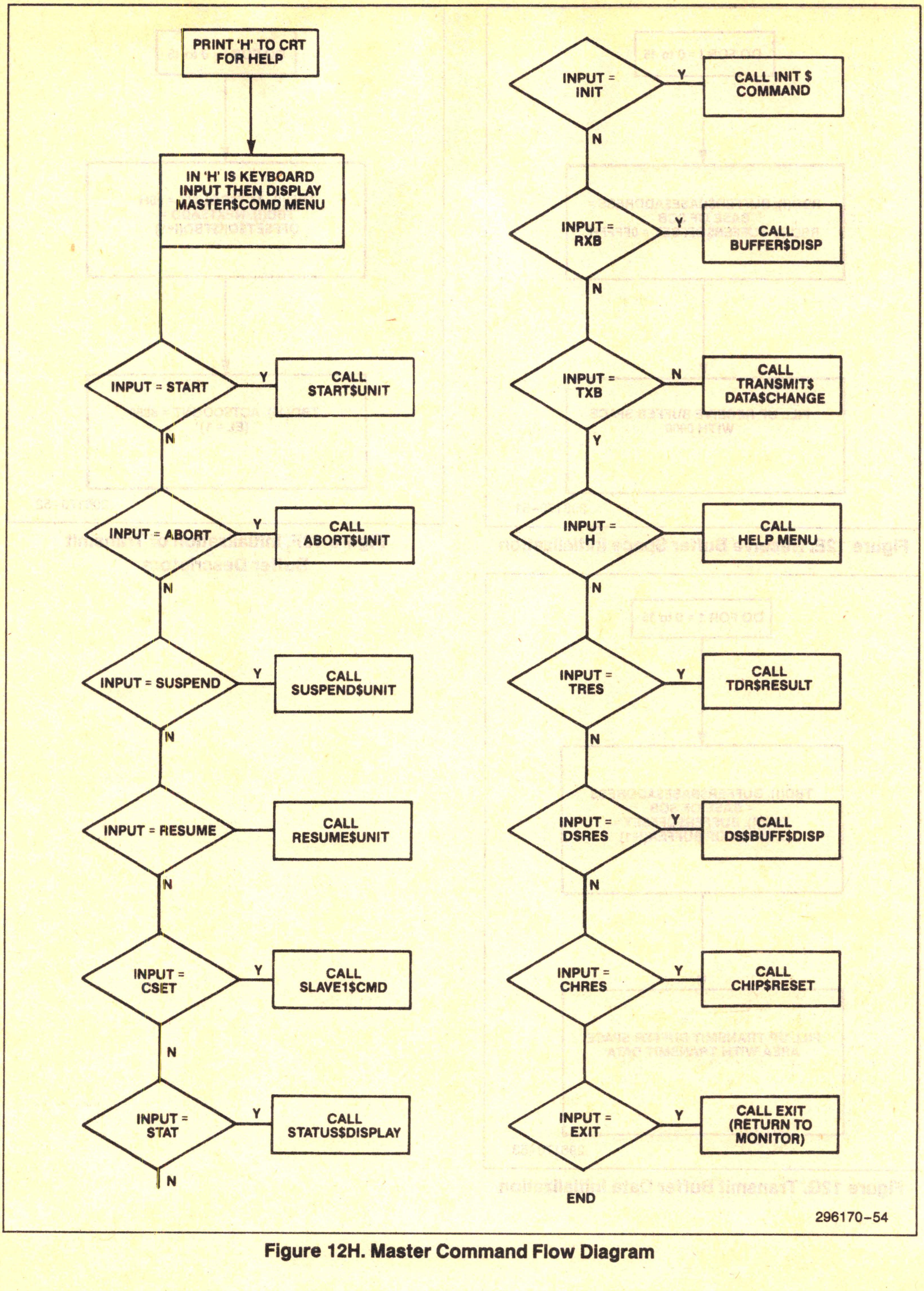


Figure 12H. Master Command Flow Diagram

'START'	: START COMMAND UNIT ON RECEIVE UNIT
'ABORT'	: ABORT CU OR RU
'SUSPEND'	: SUSPEND CU OR RU
'RESUME'	: RESUME CU OR RU
'CSET'	: COMMAND SET MENU
'STAT'	: STATUS MENU
'INIT'	: INITIALIZE COMMAND BLOCKS, : RECEIVE OR TRANSMIT BUFFERS
'RXB'	: DISPLAY RECEIVE BUFFERS
'TXB'	: DISPLAY TRANSMIT BUFFERS
'H'	: HELP
'EXIT'	: EXIT THE MONITOR
'TRES'	: DISPLAY TDR RESULT
'DSRES'	: DISPLAY DUMP STATUS BUFFER
'CHRES'	: SOFTWARE CHIP RESET

296170-55

Figure 12I. Master Command Menu

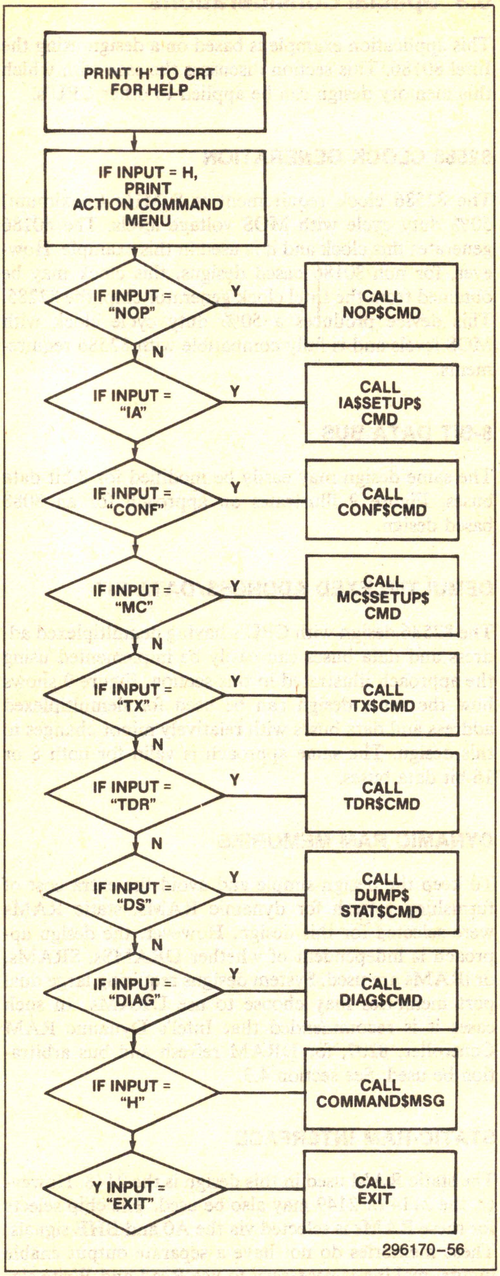


Figure 12J. Action Commands Flow Diagram

NOP	: SET UP NOP COMMAND
IA	: SET UP INDIVIDUAL ADDRESS COMMAND
CONF	: SET UP CONFIGURE COMMAND
MC	: SET UP MULTICAST COMMAND
TX	: SET UP TRANSMIT COMMAND
TDR	: SET UP TDR COMMAND
DS	: SET UP DUMP STATUS COMMAND
DIAG	: SET UP DIAGNOSE COMMAND

296170-57

Figure 12K. Action Commands Menu

6.3 Special Considerations

This application example is based on a design using the Intel 80186. This section discusses the ease with which this memory design can be applied to other CPU's.

82586 CLOCK GENERATION

The 82586 clock requirement is 8 MHz (maximum) 50% duty cycle with MOS voltage levels. The 80186 generates this clock and it is used in this example. However, for non-80186 based designs, this clock may be obtained from the Intel clock generator chip, the 82285. This device produces a 50% duty cycle clock with MOS levels and is fully compatible with 82586 requirements.

8-BIT DATA BUS

The same design may easily be modified for 8-bit data buses. Figure 9 illustrates an approach for an 8088 based design.

DEMULTIPLEXED ADDRESS/DATA BIT

The 82586 design with CPU's having demultiplexed address and data buses can easily be implemented using the approach illustrated in this section. Figure 9 shows how the same design can be used for demultiplexed address and data buses with relatively minor changes to this design. The same approach is valid for both 8 or 16-bit data buses.

DYNAMIC RAM MEMORIES

To keep the design simple and avoid the extra cost of furnishing refresh for dynamic RAMs, static RAMs were selected for this design. However, the design approach is independent of whether DRAMSs, SRAMs, or iRAMs are used. System designs requiring large dual port memories may choose to use DRAMs. In such cases it is recommended that Intel's Dynamic RAM Controller, 8207, for DRAM refresh and bus arbitration be used. See section 4.3.

STATIC-RAM INTERFACE

The static RAM used in this design is the 2148. However, the 2114 or 2149 may also be used. The chip selects for these RAMs is selected via the A0 and BHE signals. These memories do not have a separate output enable inputs, making it necessary to use Read and Write signals for chip select generation, thus avoiding bus contention between chip select and Write active.

SERIAL INTERFACE

The 82586 Serial Interface is a typical 10 Mbps Ethernet interface. The 82501 uses a 20 MHz antiresonant crystal as its clock source, and provides the Transmit and Receive clocks for the 82586.

6.4 Conclusions

The concept of dual port memory design is not new. However, the cost of such an approach has always been an issue. This design example illustrates a simple low cost design which still retains all of the advantages of the dual port design. This design may be easily modified for 8-bit CPU architectures or for systems using CPU's with demultiplexed address and data buses. This approach is suitable for system designs where the CPU bus bandwidth requirements and bus latency may retard system performance, especially when receiving serial data at 10 Mbps. This approach, while preserving the CPU bus bandwidth for other tasks, enables the user to optimize the 82586 design for very efficient bus utilization. Another advantage of this approach is the relative ease with which the 82586 can be interfaced to CPU's that are considerably different than the iAPX 186, or iAPX 86 family.

The total number of devices used for bus arbitration in this design consisted of 10–12 SSI/MSI chips. The entire circuit used up about 36 square inches of board space and the estimated cost of the board, components and connectors was less than $75, less the cost of the 82586/82501.

7.0 iNA 960 TRANSPORT ENGINE

7.1 Introduction

There are two alternatives to provide a system with communications capability. One configuration is to run the communications software as a job under the host operating system. In this implementation, the communication software shares host processor resources with other user tasks, see Figure 13. The second configuration is to run the communications software on a dedicated processor. In this implementation, the communication software runs separately from the host and other user programs. This latter implementation is frequently called a Communications Front End Processor, or Comm-Engine. When the Comm-Engine is implementing the Transport Layer function, it is called a Transport Engine, see Figure 14.

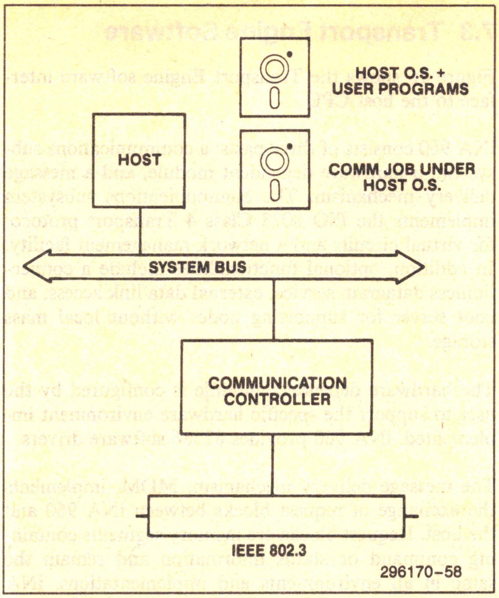


Figure 13. Communications Software Running on a Host

The Comm-Engine approach provides system designers with three important benefits: performance, flexibility and easy system upgrades. First, the Comm-Engine affords high performance because it offloads the host from supporting communication tasks. Thus, high communication throughput can be realized because the CPU is not supporting other user programs. Second, the Comm-Engine can be used to contribute to systems because it fits into a modular system architecture. Modular systems are attractive because they facilitate future system expansion and upgrades. Modularity also simplifies service and maintenance of the system. Third, the Comm-Engine will enable existing systems to be upgraded to embody networking capability, without degrading system performance.

The Intel 82586 LAN Coprocessor, a dedicated iAPX 86/88/186 processor and iNA 960 Network Software are the building blocks required to realize a Transport Engine. This Transport Engine provides the user with ISO standard compatible Transport capabilities in any hardware and software system environment.

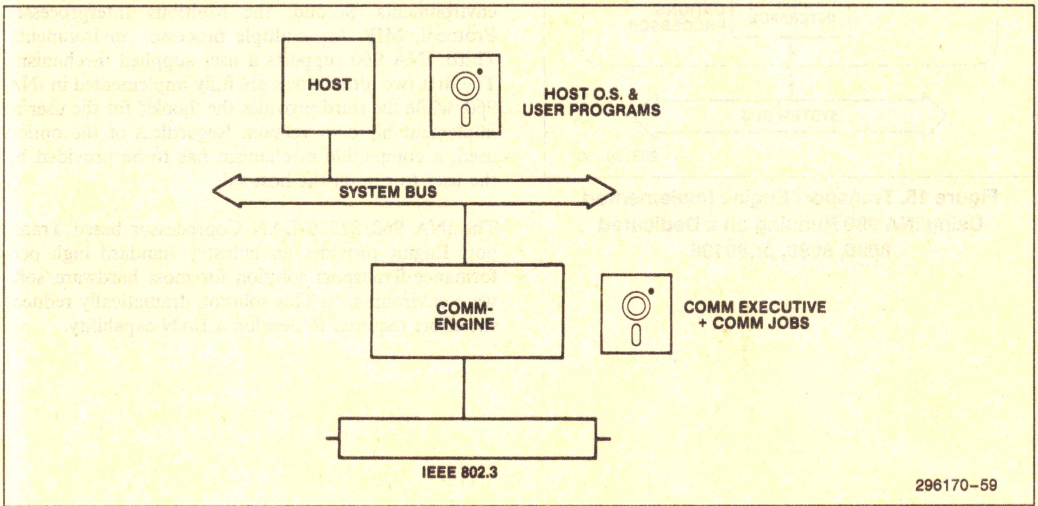


Figure 14. Communications Front End Processor, Comm-Engine

7.2 Transport Engine Hardware

Figure 15 shows the block diagram of a generic Transport Engine based on the 82586. The 80186 processor is used to eliminate the need for 'TTL glue,' and thus be the lower cost implementation.

In addition to the processor, the Transport Engine requires local RAM for data buffering and ROM to store communication software. A typical configuration requires 30 to 50K bytes of ROM for iNA 960, and 16 to 64K bytes of RAM for data buffering and supporting the software.

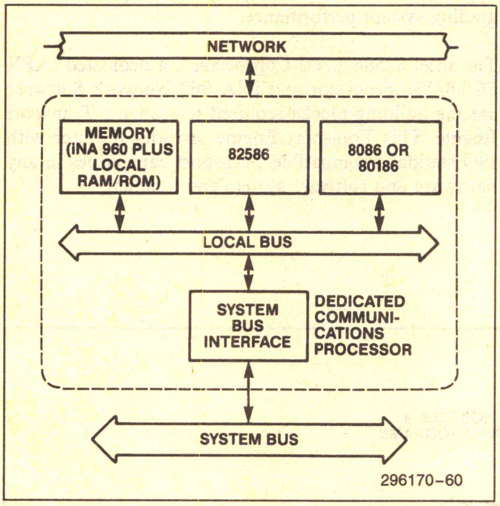


Figure 15. Transport Engine Implemented Using iNA 960 Running on a Dedicated 8086, 8088, or 80186

7.3 Transport Engine Software

Figure 16 shows the Transport Engine software interface to the host CPU.

iNA 960 consists of three parts: a communications subsystem, a hardware dependent module, and a message delivery mechanism. The communications subsystem implements the ISO 8073 Class 4 Transport protocol for virtual circuits and a network management facility. In addition, optional functionalities include a connectionless datagram service, external data link access, and boot server for supporting nodes without local mass storage.

The hardware dependent module is configured by the user to support the specific hardware environment implemented. iNA 960 provides 82586 software drivers.

The message delivery mechanism, MDM, implements the exchange of request blocks between iNA 960 and the host. Request blocks are memory segments containing command or status information and remain the same in all environments and implementations. iNA 960 provides the user with three choices of message delivery mechanisms. First, the Base Control Block interface mechanism to interface iNA 960 to single host environments. Second, the Multibus Interprocessor Protocol, MIP, for multiple processor environments. Third, iNA 960 supports a user supplied mechanism. The first two alternatives are fully implemented in iNA 960, while the third provides the 'hooks' for the user to implement his own version. Regardless of the option used, a compatible mechanism has to be provided by the user to run on the host side.

The iNA 960/82586 LAN Coprocessor based Transport Engine provides an industry standard high performance Transport solution for most hardware/software environments. This solution dramatically reduces the effort required to develop a LAN capability.

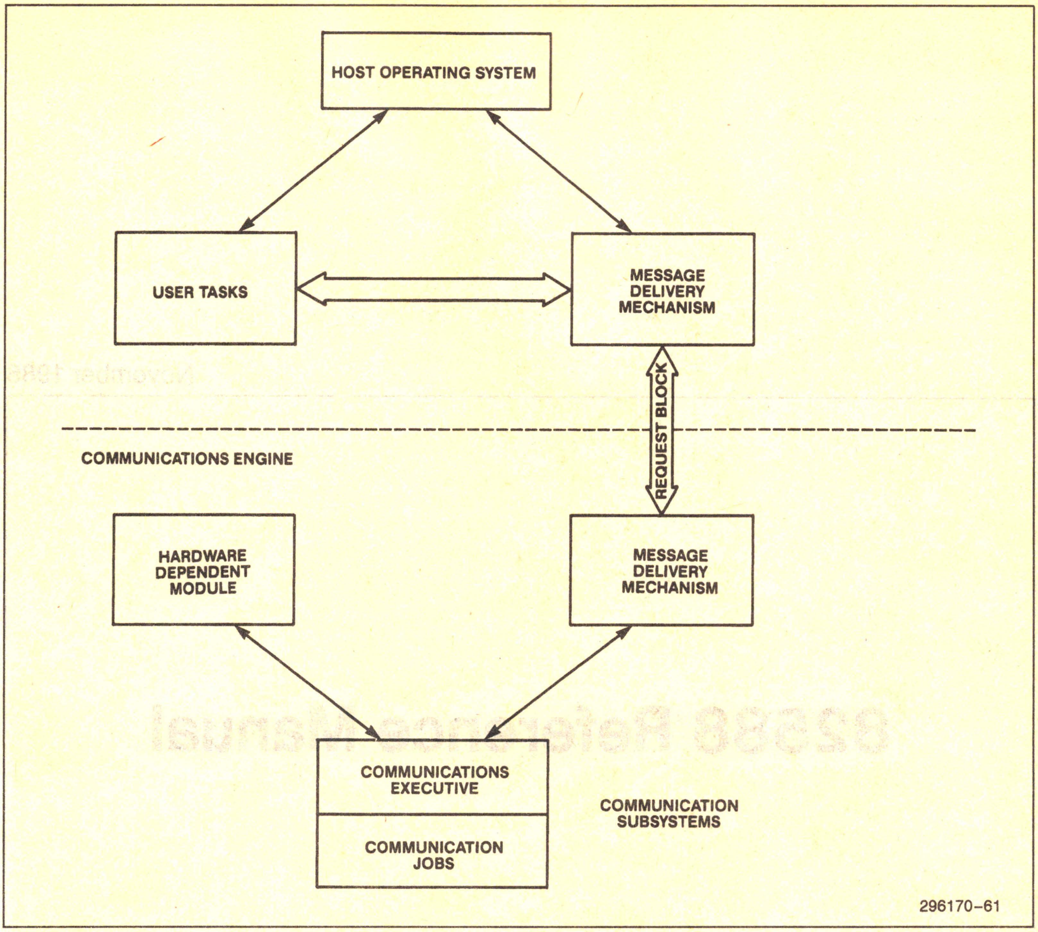


Figure 16. The Comm-Engine/Host CPU Software Interface

intel®

November 1986

82588 Reference Manual

Order Number: 231368-003

1.0 INTRODUCTION

The 82588 is a highly integrated Local Area Network (LAN) Controller that lends itself easily to cost sensitive LAN applications such as Personal Computers and Intelligent Terminals. Because of its high integration, the 82588 reduces component count which in turn reduces board space and development time. Additionally, due to the flexibility of the 82588, the system design engineer can program a variety of device parameters to a configuration that allows it to be used in the emerging IEEE 802.3 standards for PCs. The combination of low cost and high flexibility makes the 82588 an attractive solution for the LAN system designer.

The 82588 provides most of the functions of the ISO Physical and Data Link layers of LAN architecture. This includes a CSMA/CD controller, two different collision detect mechanisms, and a data encoder/decoder that supports data transfer rates of up to 2 Mbps.

As mentioned, the 82588 is programmable which allows it to operate in a variety of LAN environments including the emerging IEEE 802.3 standard of 1 Mbps baseband (called STARLAN) and the IBM PC Network 2 Mbps broadband. Some of the programmable parameters are:

— Framing (End of Carrier or SDLC)

— Address Field Length

— Station Priority

— Interframe Spacing

— Slot Time

— CRC-32 or CRC-16

— NRZI or Manchester encoding/decoding

In addition, the 82588 allows for a variety of frame formats, network topologies, data encoding and decoding schemes and data transfer rates.

A major innovation of the 82588 is its on-chip logic based collision detection. So that a high probability of collision detection can be maintained, two mechanisms are provided. First, the Code Violation method defines a collision when a transition edge occurs outside of the region specified for NRZI or Manchester encoding. Second, the Bit Comparison method compares the signature of a transmitted frame to the receive frame signature while "listening to itself."

The 82588 goes beyond the basic provisions required for LAN physical interfacing. It also has an extremely friendly system interface that makes it easy to design with. The 82588 has a high level command interface (the CPU sends commands such as CONFIGURE or TRANSMIT) so the designer does not have to bother with low level software development. This saves on CPU overhead as well. The 82588 makes efficient use of the system memory available by employing Multiple Buffer Reception in which receive frames are saved in buffers that are chained together in system memory. This is an important feature for those applications with limited memory such as personal computers. The 82588 has two independent 16 byte FIFOs (one for reception and one for transmission) that allows the device to tolerate bus latency. Finally, the 82588 provides an 8-bit data path that supports up to 4 Mbytes/second using external DMA. The DMA interface has been optimized for use with the 80186/80188 microprocessors.

The 82588 provides a rich set of diagnostic and network management functions that allow the designer to minimize debug time and maintain top network efficiency. Specifically, the 82588 provides internal and external loopback capability and network channel activity indicators, it can capture all frames regardless of address (Promiscuous Mode) and it supports Time Domain Reflectometry for locating shorts and opens in the transmission cable. There is also a Register Dump command which allows the user to examine the 82588 internal status registers.

The following chapters describe the features and capabilities of the 82588 in greater detail.

2.0 THE 82588 INTERNAL ARCHITECTURE

There are two major functional blocks of the 82588: a parallel system interface to the host CPU and a serial interface to the local area network. Linking these functional blocks are two on-chip, 16-byte FIFOs, one each for transmitting and receiving (see Figure 1).

2.1 Parallel Section

The parallel system interface consists of a Bus Interface Unit (BIU) and several registers. The BIU has an 8-bit data bus, and is responsible for all interfacing to the system bus. It handles all transfer of data to and from memory — at speeds of up to 4 Mbytes/second — as well as issuing CPU commands and interrupts. There are two DMA channels allowing for simultaneous transmission and reception. The register section consists of three register sets. The first stores configuration information, the second is for the posting of commands and the third contains status information.

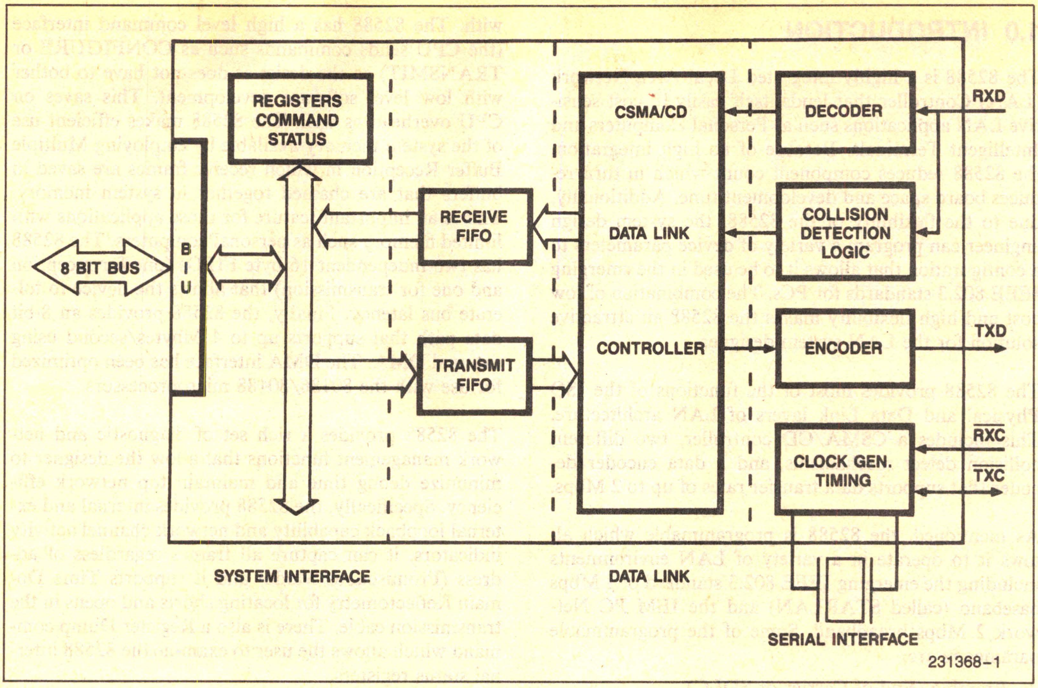


Figure 1. 82588 Block Diagram

2.2 Serial Section

The serial machine of the 82588 employs the IEEE 802.3 CSMA/CD protocol. The serial section is responsible for the following tasks:

— converting data from parallel to serial form and vice versa

— formatting the frame per the programmed configuration

— computing the CRC "signature" and monitoring for code violations in order to detect collisions

— performing carrier sense and deference if the network is busy

— calculating random delay time before retransmission in the event of deference or collision

The efficiency of CSMA/CD was demonstrated in a 1979 study of an Ethernet LAN of about 120 host computers and network server devices. The study showed that a network using collision detection has a throughput rate of 98 percent. This can be compared to the efficiency rate of 37 percent in networks with collision avoidance and 18 percent in networks with no collision regulating provisions. Until recently, the relative sophistication of collision detection schemes such as CSMA/CD prevented their inclusion in cost effective LAN components.

The data encoder/decoder of the 82588 can transmit and receive data in any one of three formats: Manchester, differential Manchester and NRZI encoding. Data rates with the internal encoder/decoder, can be up to 2 Mbs. The serial machine is driven by a clock generator using an up to 16 MHz crystal. This clock generator is for the serial side only; the parallel side receives its timing from the system clock.

COLLISION DETECTION

A major breakthrough introduced in the 82588 is the on-chip logic based collision detection capability. There are two programmable forms of collision detection. The "code violation" detection method checks the incoming bits to see if they violate Manchester or NRZI standards. If a violation does occur, the 82588 assumes a collision has occurred. The second method is called "bit comparison." For this method the 82588 listens to itself while calculating a signature for both the transmit and receive data. If the signatures do not match, the 82588 will back-off and retry immediately, without having to wait for transmission of the frame to be completed. In this way, total data throughput is increased significantly.

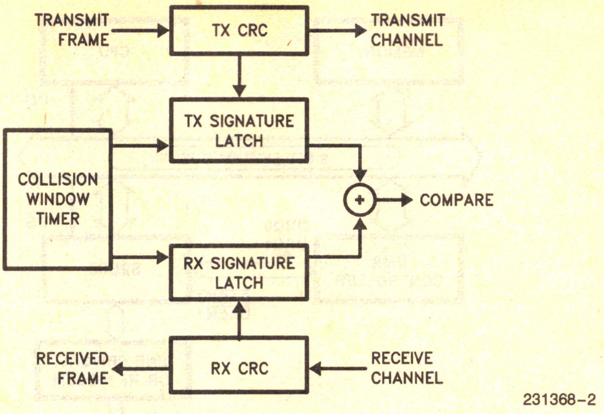


Figure 2. Bit Comparison Detect

The parallel and serial section of the 82588 are asynchronous. The interface between them is made up of two unique FIFOs, one for transmission and one for reception. The FIFOs are 16 bytes deep and improve the data transfer rate in two ways. First, DMA requests to the CPU are minimized by fine tuning the programmable FIFO threshold to match the particular system bus latencies. Second, the CPU side of the chip does not have to wait for the network side which transfers data at a slower rate.

3.0 WORKING WITH THE 82588

This section describes how the 82588 interacts with system CPU, and how the 82588 transmits and receives frames. The emphasis here is not on any particular commands, but rather on how the 82588 works. The details on Framing, Network Management, Initializing the 82588, Configuring the 82588, Controlling the 82588, System Interface, and Link Interface are contained in the following chapters.

3.1 82588/Host CPU Interaction

The CPU communicates with the 82588 through the System's memory and 82588's on-chip registers. The CPU creates a data structure in the memory, programs the external DMA controller with the start address and byte count of the block, and issues the command to the 82588.

The 82588 is optimized for operating with the iAPX 186/188, but due to the very conventional nature of hardware signals between the 82588 and the CPU, the 82588 can operate easily with other processors. The data bus is 8 bits wide and there is no address bus.

Chip select and Interrupt lines are used to communicate between the 82588 and the host as shown in Figure 3. Interrupt is used by the 82588 to draw the CPU's attention. The Chip Select is used by the CPU to draw the 82588's attention.

There are two kinds of transfers over the bus: Command/Status and data transfers. Command/Status transfers are always performed by the CPU. Data Transfers are requested by the 82588, and are typically performed by a DMA controller.

The CPU writes to 82588 using $\overline{CS}$ and $\overline{WR}$ signals. The CPU reads the 82588 status register using $\overline{CS}$ and $\overline{RD}$ signals.

To initiate a command like Transmit or Configure, a Write operation to 82588 is issued by the CPU. A Read operation from CPU gives the status of the 82588. Although there are four status registers, they are read using the same port in a round robin fashion. Section Eight discusses details on these commands, and the status registers.

Any parameters or data associated with the command are transferred between the memory and 82588 using DMA. The 82588 has two data channels, each having Request and Acknowledge line. Typically one channel is issued to receive data and the other to transmit data and to do all the other initialization and maintenance operations like Configure, Address Set-Up, Dump, Diagnose, etc. The two channels are identical and can be used interchangeably.

When 82588 requires access to the memory for parameter or data transfer, it activates the DMA request line and uses the DMA controller to achieve the data transfer. Upon the completion of an operation, the 82588 interrupts the CPU. The CPU then reads results of the operation and the status of the 82588.

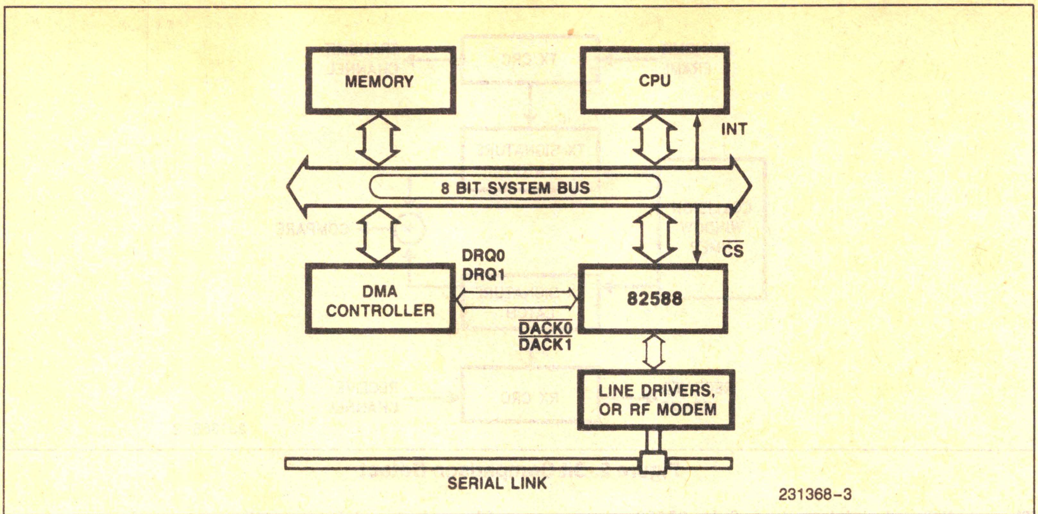


Figure 3. 82588/Host CPU Interaction

3.2 Transmitting a Frame

To transmit a frame, the CPU prepares a Transmit Data Block in memory as shown in Figure 4. Its first two bytes specify the length of the rest of the block. The next few bytes (up to 6 bytes long) contain the destination address of the node it is being sent to. The rest of the block is the data field. The CPU programs the DMA controller with the start address of the block, length of the block and other control information and then issues the Transmit command to the 82588. (Section 4 discusses details on the framing).

Upon receiving the command, the 82588 fetches the first two bytes of the block using DMA to determine the length of the block. If the link is free, the 82588 begins transmitting the preamble and concurrently fetches the bytes from the Transmit Data Block and loads them into a 16 byte FIFO to keep them ready for transmitting. The FIFO is a buffer between the serial and parallel part of the 82588. If the link is busy, the 82588 fills up the transmit FIFO and defers. The on-chip FIFOs help the 82588 to tolerate system bus latency as well as provide efficient usage of system bus bandwidth.

The destination address is sent out after the preamble. This is followed by the source or the station individual address, which is stored earlier on the 82588 using the IA-SETUP command. After that, the entire information field is transmitted followed by a CRC field calcu-

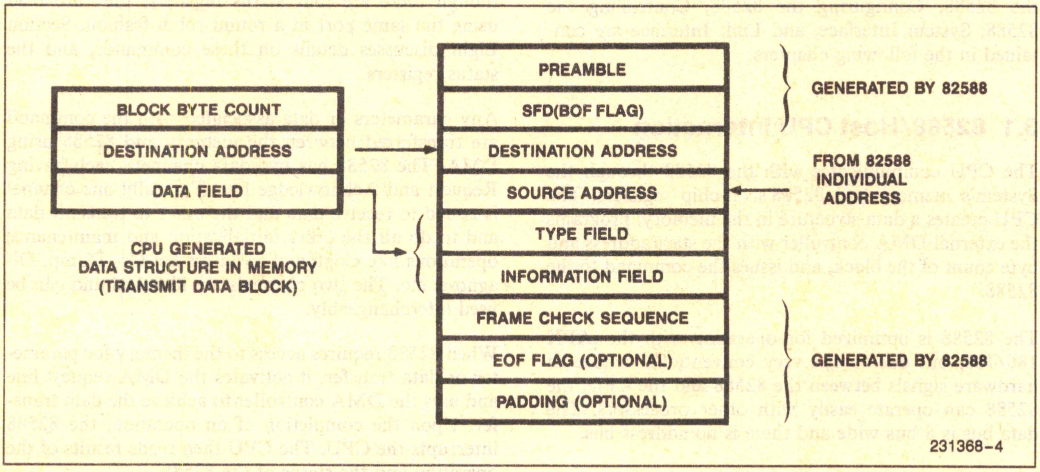


Figure 4. The 82288 Frame Structure and Location of Data Element in System Memory

lated by the 82588. If during the transmission of the frame, a collision is encountered, then the transmission is aborted and a jam pattern is sent out after completion of the preamble. The 82588 generates an Interrupt indicating the experience of a collision and the frame has to be retransmitted. Retransmission is done by the CPU exactly as the TRANSMIT command except the RETRANSMIT command keeps track of the number of collisions encountered. When the 82588 gets the Retransmit command and the exponential back-off time is expired, the 82588 attempts transmitting the frame again. The transmitted frame can be coded to either Manchester, Differential Manchester or NRZI methods.

3.3 Receiving a Frame

The 82588 can receive a frame when its receiver has been enabled. The received frame is decoded by either on-chip Manchester, Differential Manchester or NRZI decoders in High Integration Mode or by external decoders in the High Speed Mode. The 82588 checks for an address match for an Individual address, a Multicast address or a Broadcast address. In the Promiscuous mode the 82588 receives all frames. Only when the address match is successful does the 82588 transfer the frame to the memory using the DMA controller. Before enabling the receiver, the CPU makes a memory buffer area available to the Receive Unit, and programs the starting address of the buffer in the DMA controller. The received frame is transferred to the memory buffer.

4.0 FRAMING AND LINK MANAGEMENT

The data structures handled by the 82588 are called frames. A frame is a sequence of bits that travels on the link.

Framing has three primary functions: to determine the beginning and the end of the frame (frame boundary delineation); to determine the frame's source and destination (addressing); and to perform transmission error detection.

4.1 Frame Format

The Frame format supported by the 82588 is shown in Figure 5.

PREAMBLE
SFD (BOF FLAG)
DESTINATION ADDRESS
SOURCE ADDRESS
INFORMATION FIELD
FRAME CHECK SEQUENCE
EOF FLAG (OPTIONAL)
PADDING (OPTIONAL)

Figure 5. 82588 Generalized Frame Format

The different fields of the frame are:

— The Preamble which is used as a synchronizing sequence for bit decoding.

— SFD, Start of Frame Delimiter (for bitstuffing method, this field is called BOF flag).

— The Destination Address (frame target address).

— The Source Address (sender's address).

— The Information field containing user supplied data.*

— The Frame Check Sequence (FCS): Cyclic Redundancy Check (CRC) used in detecting bit errors.

— The End of Frame (EOF) flag - optional. (For bitstuffing mode.)

— Padding: optional field which extends the length of the frame to ensure minimum frame length. Optional (for bitstuffing mode).

* The 802.3 Length Field is included in the Information field.

4.2 Frame Boundary Delineation

The 82588 handles Frame Boundary Delineation transparently to the user. The fields involved in Frame boundary delineation are: Preamble, SFD (BOF Flag), EOF flag and Padding. The 82588 is configurable to one of two Frame delineation methods: End of Carrier (802.3 compatible) or Bitstuffing.

In the End of Carrier method, the 82588 transmits (depending on the configuration) 1, 3, 7, 15 preamble bytes of alternating ones and zeros followed by a SFD (BOF Flag) of pattern 10101011. The end of frame is indicated by the carrier going inactive immediately after the Frame Check Sequence field. The Carrier is a signal which informs the 82588 of activity on the link. It can be internally or externally generated. This frame boundary delineation method is compatible with IEEE 802.3.

The Bitstuffing method implements the HDLC zero bit insertion/deletion mechanism. The 82588 transmits (depending on the configuration) a Preamble of 1, 3, 7 or 15 bytes of alternating ones and zeroes followed by an HDLC flag (01111110). End of frame is indicated by a HDLC flag. The 82588 can perform HDLC zero bit insertion (insert 0 after five consecutive 1's) on the fields between the BOF and EOF flags. The chip can be configured to pad the frame with additional flags so that the frame length becomes equal to or longer than the Slot Time.

4.3 Addressing

Regardless of the Frame Boundary Delineation method, the two fields following the SFD (BOF Flag) are Destination Address and Source Address.

Addressing allows frames to be directed to one or more specific nodes. The 82588 provides flexible addressing techniques allowing a frame to be received by a single node, a group of nodes (multicast addressing), or all nodes (broadcast addressing).

After initialization, the 82588 is configured with an Individual Address using the IA-SETUP command. The address length is determined by the Address Length parameter (0 to 6 bytes). The default configuration is an all ones address (Broadcast Address). 6 bytes long.

During transmission the 82588 usually inserts its Individual Address from the internal Individual Address register into the Source Address field. The source address insertion can be overridden when configuring the chip. During reception, the 82588 compares the incoming Destination Address with its Individual Address, the programmed multicast set of address and the broadcast address. If an address match is made (all bits must be equal) the frame is accepted. A frame whose address does not match is simply ignored by the 82588 and has no effect on the 82588 nor on any other component of the station.

INDIVIDUAL ADDRESS

The 82588 is normally configured with a specific Individual Address. An Individual Address must have a zero in the least significant bit.

BROADCAST ADDRESSING

An address with all 1's is the Broadcast Address. Every station on the link "hears" a Broadcast. A frame may be targetted to all nodes by using the broadcast address. The 82588 can be configured to disable reception of frames with Broadcast Destination Address, e.g., for stations with limited storage resources.

MULTICAST ADDRESSING

Frames can be directed to a specific group of nodes. This group can have a particular group address called Multicast Address. A node may belong to several different groups. A one in the least significant bit of the destination address distinguishes Multicast Address from Individual Address. MC-SETUP command is used to program the set of Multicast addresses for a station by setting up a 64 bit "Hash Table." The 82588 maps and stores every Multicast Address into a single bit of the Hash table. During reception of a frame whose Destination Address is a Multicast Address, the 82588 checks whether this address is mapped in the Hash table. If an address match is determined, the 82588 begins the reception process.

It is possible for more than one Multicast Address to be mapped into a given Hash bit. Thus, the host may have to perform additional checking. If 64 or fewer Multicast addresses are used in a system, it is possible to select address values that map into unique bits in the Hash table. The Hashing function is the CRC polynomial used for bit error detection. The 6 most significant bits (2–7) are selected from the first byte of the CRC shift register to index the 64-bit Hash table.

INFORMATION FIELD

The information field follows the source address field. It contains the actual data being transferred in the frame. Its maximum length is $(2^{16}) - 2 \times$ (address length) $- 2$ which includes destination and source addresses.

4.4 Error Detection

The Frame Check Sequence (FCS) Field protects against bit errors in the frame. It is the result of a Cyclic Redundancy Check computed on the Destination Address, Source Address, and Information Fields. The chip can be configured to one of two CRC algorithms: the CCITT V-41 (HDLC) 16-bit polynomial or the Autodin II (IEEE 802.3) 32-bit polynomial.

The CRC mechanism is transparent to the user. During transmission, the 82588 calculates and inserts the Frame Check Sequence. During reception the chip verifies the correctness of the incoming Frame but does not pass the FCS to memory. The CRC is applied to an integral number of 8-bit words excluding the potential residue. CRC insertion can be disabled for diagnostic purposes. The user should then provide the FCS field after the data field.

4.5 Frame Transmission

The 82588 transfers data from host memory to the network when a CPU issues a TRANSMIT command. Before instructing the 82588 to start a transmission, the host CPU prepares the frame in memory. Parts of the frame such as Preamble Source Address, CRC and Flags are inserted by the 82588 Data Link Controller. The 82588 resolves access and contention on the link using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) link Management protocol. When the transmission is completed, the 82588 updates a set of status registers and raises its Interrupt signal to inform the CPU.

4.6 Link Management

The 82588 handles CSMA/CD link management algorithms according to the IEEE 802.3 standard. 82588 link management algorithms are adaptable to a variety of network topologies via programmable configuration parameters. Station priorities are also programmable.

The 82588 constantly monitors link activity. Whenever it senses activity on the link the 82588 defers to the passing frame by delaying any pending transmission. When there is no more activity on the link, the 82588 continues to defer for Inter Frame Spacing (IFS) time (minimum time between two consecutive frame transmissions). This parameter is configurable from 12 to 255 bit times. If at the end of that time, the 82588 has a frame waiting to be transmitted, transmission is initiated independently of the link activity.

Once the 82588 has finished deferring and has started transmission, it is possible to experience link contention. This situation is called a collision and is generally detected and signalled by the transceiver. The 82588 can be programmed to detect collisions internally or externally via the CDT pin. When the 82588 experiences a collision, it enforces the collision by transmitting a jam pattern of 32 to 48 bits.

If a collision is detected during the Preamble, transmission of the preamble is completed before jamming starts. The collision enforcement mechanism ensures detection of the collision by all the stations.

The jam pattern is all ones in all configurations except for NRZI encoding, where it is a sequence of all zeros. (Transition every bit time).

The dynamics of collision handling are largely determined by the Slot Time. Slot Time is the maximum end to end round trip delay time of the network plus jam time. Slot Time is important because it is the worst case time to detect a collision. The Slot Time is programmable from 1 to 2048 bit times.

The 82588 notifies the user of a collision and gives an opportunity to retransmit at the end of the Backoff time out.

The 82588 implements the IEEE 802.3 scheduling of the retransmission. The controlled randomization process is called "truncated binary exponential backoff." A retransmission is delayed an integral multiple of slot times. The number of slot times to delay before the nth retransmission attempt is chosen as a uniformly distributed random integer in the range $0 \leq r < 2^k$, where $K = \min [n, 10]$. The number of retransmission attempts is programmable in the range 0 to 15. The beginning of the Backoff time is configurable to one of two methods: If configured to the IEEE 802.3 compatible method, Backoff starts immediately after the end of the Jam pattern; if configured to the alternate Backoff method (designed for lower bit rates and/or shorter topologies), Backoff starts after the deferring period following collision.

After the backoff time has expired and the CPU has issued a RETRANSMIT command, the 82588 attempts to retransmit the frame (after deferring to any traffic on the link) unless the number of retransmissions has exceeded the maximum allowed.

The 82588 maintains a retry counter that is incremented after each collision. If retransmission is successful, the user is notified. If the number of retries equals the programmed maximum, an error is reported. The number of allowed retries is configurable from 0 to 15 attempts. The only difference between transmission and retransmission is that transmission clears the retry counter and retransmission increments it.

On completion of transmission and retransmission, the 82588 reports the total number of collisions that have occurred and whether it equalled or exceeded the maximum. It also indicates if the chip had to defer to passing traffic on its first transmission attempt.

After transmission has started the 82588 attempts to transmit the entire frame. In the normal case, frame transmission is completed and the host is notified through interrupt and status register. Otherwise, one or more of the following events causes transmission to terminate prematurely: Clear-to-Send signal goes inactive during transmission, data transfer rate from memory to the chip did not keep up with transmission (DMA underrun), Carrier Sense goes inactive during transmission (if the chip is in the mode where carrier sense is expected during transmission), a collision is detected via Collision Detect Signal. These events are also reported to the CPU via the status registers.

The user may attempt to abort transmissions. Upon receipt of the ABORT command, the 82588 transmits a Jam pattern to cause a CRC mismatch. The chip reports to the host either that the ABORT succeeded or that the frame transmission was completed before the ABORT was accepted.

4.7 Priority Mechanism

The 82588 implements two different priority mechanisms: linear and accelerated contention resolution. Either may be used to distribute relative priority among stations.

Linear priority determines the number of Slot Times that the 82588 waits after deferring or after the end of the Backoff Time (whichever occurs last) before transmitting. If the link becomes busy during the wait period, the process of deferring and waiting starts again. The Linear priority is programmable from 0 to 7. Zero provides the highest priority and is IEEE 802.3 compatible.

Accelerated contention resolution extends the range from which the random number for backoff is drawn. The random number for the first Backoff delay is in the range $0 \leq r < 2^P$, where P is configurable from 0 to 7 with 0 providing the highest priority.

4.8 Frame Reception

The 82588 receives and passes to memory every frame whose address matches its individual, multicast or broadcast address. By configuring the chip to the promiscuous mode, it will receive and pass to memory all frames regardless of address.

The 82588 constantly monitors the serial link activity. When the link becomes active, the 82588 starts accepting the incoming bits.

The Preamble and SFD (BOF flag) are discarded. The 82588 compares the incoming frame's Destination Address to its own Individual Address. If there is an address match and the frame length equals or exceeds 6 bytes, the 82588 passes the Destination Address, Source Address, and Information fields to system memory via the DMA controller. The 82588 verifies correctness during reception of the FCS field. If there is no address match, the 82588 does not request DMA controller's attention and becomes ready to receive the next frame.

When the reception is completed, the 82588 updates a set of status registers and raises its Interrupt signal to inform the CPU. If the received frame has errors (CRC violation, alignment error, DMA overrun), the CPU can reclaim the memory used to store that frame. The next received frame can then be stored in the reclaimed memory.

When configured to IEEE 802.3 End of Carrier delineation, end of frame is indicated by the carrier going inactive. The number of bits after the BOF flag must be a multiple of eight. Residual "dribble" bits are discarded, and not included in the Frame Check Sequence. An alignment error is reported when the frame is "misaligned" (has got residual bits) and a CRC error is detected.

When configured to Bitstuffing delineation, the 82588 performs zero bit deletion, discards the EOF flag, and all following bits until end of carrier. Residual bits are discarded in a manner similar to the End of Carrier method. An error is reported if the carrier goes inactive prior to recognition of an EOF flag.

The minimum frame length is configurable in the range from 6 to 255 bytes. Any frame containing less than the minimum (configured) number of bytes is presumed to be a fragment resulting from a collision or an aborted transmission. Any frame shorter than 6 bytes is discarded without notifying the user.

SINGLE BUFFER MODE

The received frame is transferred to the memory buffer in the format shown in Figure 6. This method of reception is called "Single Buffer" reception. The entire frame is contained in one continuous buffer. Upon completion of reception the total number of bytes written into the memory buffer is loaded into status registers 1 and 2, and the status of the reception itself is appended to the received frame. An interrupt to the CPU follows.

MULTIPLE BUFFER MODE

If the frame size is unknown, memory usage is optimized by using "Multiple Buffer" reception. The frame, as it comes in, is deposited into a series of fixed size buffers. This way the user does not have to allocate large memory space for the short frames. Instead, the 82588 can dynamically allocate memory space as it receives frames. This method requires both data channels alternately to receive the frame. As the frame reception starts, the 82588 interrupts the CPU and automatically requests assignment of the next sequential buffer. The CPU does this and loads the second DMA channel with the next buffer information so that the 82588 can immediately switch to the other channel as soon as the current buffer is full. When the 82588 switches from the first to the second buffer it again interrupts the CPU requesting it to allocate another buffer on the other (previous) channel in advance. This process continues until the entire frame is received. The received frame is spread over multiple memory buffers. The link between the buffers is easily maintained by the CPU using a buffer chain descriptor structure in memory (see Figure 7).

This dynamic (pre) allocation of memory buffers results in efficient use of available storage when handling frames of widely varying sizes. Since the buffers are pre-allocated one block in advance, the system is not time critical.

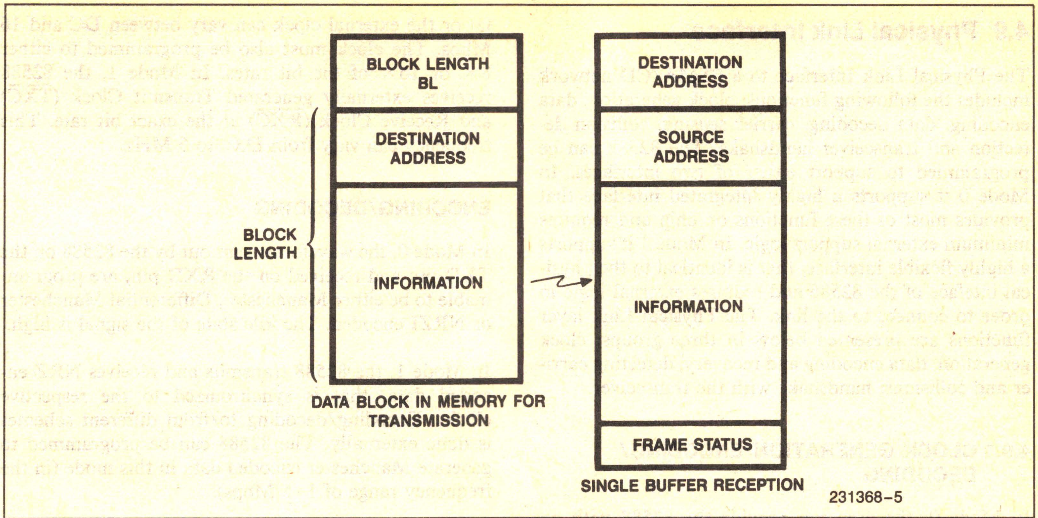


Figure 6. Single Buffer Reception

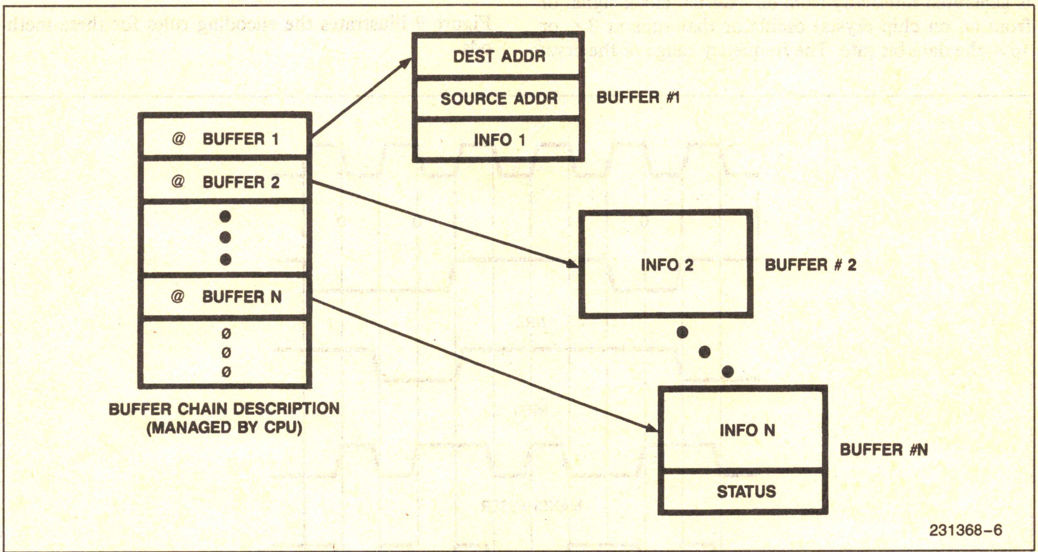


Figure 7. Multiple Buffer Reception

4.9 Physical Link Interface

The Physical Link Interface to a CSMA/CD network includes the following functions: clock generation, data encoding, data decoding, carrier sensing, collision detection and transceiver handshake. The 82588 can be programmed to support either of two interfaces. In Mode 0 it supports a highly integrated interface that provides most of these functions on chip and requires minimum external support logic. In Mode 1 it supports a highly flexible interface, that is identical to the physical inteface of the 82586 and requires external logic in order to connect to the link. The Physical Link layer functions are presented below in three groups: clock generation, data encoding and recovery; detecting carrier and collisions; handshake with the transceiver.

4.9.1 CLOCK GENERATION/ENCODING/DECODING

In Mode 0, the user can provide the 82588 with an external sampling clock or provide it with a crystal and let the 82588 generate the sampling clock. The bit clock is generated internally from an external clock signal or from an on chip crystal oscillator that runs at 8× or 16× the data bit rate. The frequency range of the crystal or the external clock can vary between DC and 16 Mbps. The clock must also be programmed to either 8× or 16× of the bit rates. In Mode 1, the 82588 receives externally generated Transmit Clock ($\overline{TXC}$) and Receive Clock ($\overline{RXC}$) at the exact bit rate. This frequency can vary from D.C. to 5 MHz.

ENCODING/DECODING

In Mode 0, the waveforms sent out by the 82588 on the TXD pin and received on the RXD pin, are programmable to be either Manchester, Differential Manchester or NRZI encoded. The idle state of the signal is high.

In Mode 1, the 82588 transmits and receives NRZ encoded data, that is synchronized to the respective clocks. Encoding/decoding to/from different schemes is done externally. The 82588 can be programmed to generate Manchester encoded data in this mode (in the frequency range of 1–5 Mbps).

Figure 8 depicts various kinds of data encoding methods implemented in the 82588.

Figure 9 illustrates the encoding rules for these methods.

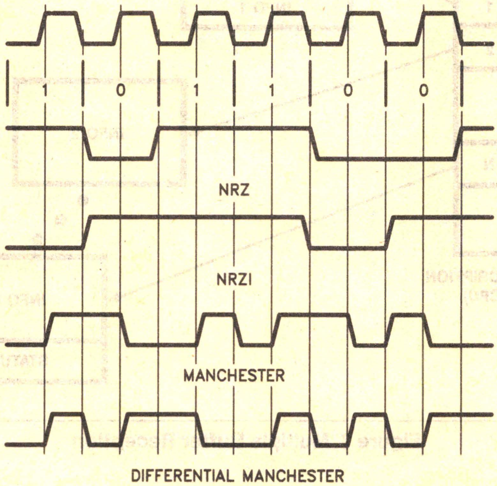


Figure 8. 82588 Data Encoding Methods

Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
NRZ	do not exist	identical to original data
NRZI	do not exist	exist only if original data bit equals 0. Dependent on present encoded signal level: to 0 if 1 to 1 if 0
MANCHESTER	exist for every bit of the original data: from 0 to 1 for 1 from 1 to 0 for 0	exist for consequent equal bits of original data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
DIFFERENTIAL MANCHESTER	exist for every bit of the original data. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0	exist only if original data bit equals 0. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0

Figure 9. 82588 Data Encoding Rules

4.9.2 CARRIER SENSE/COLLISION DETECT

CARRIER SENSE

Carrier Sense indicates that there is activity on the link, i.e., the signal from a transmitting station has reached the station that is checking.

In Mode 0, Carrier Sense signal is generated internally from the incoming data. In this mode carrier sense is consider active after detection of 3 consecutive edges. Carrier is considered inactive after:

DIFF/MANCHESTER -1.5 bit times of high level.

NRZI -8 bit times of high level.

In Mode 1, the 82588 can be programmed to either generate Carrier Sense internally (for transceivers that generate RXC only during actual reception) or accept it from the outside, via the CRS# pin. The Carrier Sense may undergo noise filtering, i.e., it must be present for a programmable number of bit times before it is considered valid.

COLLISION DETECTION

Collision Detect indicates that two or more stations are transmitting simultaneously. When a collision is detected during transmission, the 82588 jams the link, stops transmitting and goes into backoff. Jamming will not start unless preamble transmission is first completed. Collisions are detected during receptions, causing a receive frame abortion. (Only when not in external loopback mode). The 82588 can be programmed to detect collisions internally or to accept it from the outside, via the CDT# pin. The Collision Detect signal may undergo filtering, similar to Carrier Sense.

COLLISION DETECTION BY CODE VIOLATION

In Mode 0, a collision is detected internally when the receive data experiences code violations (Manchester, Differential Manchester or NRZI) while the station is transmitting.

A code violation occurs if any of the following is detected:

×8 Manchester - Pulse width:
1/8 to 2/8 or 11/8 to 12/8
"0" level for 13/8 or more
Missing mid-bit-cell transition

×16 Manchester - Pulse width:
1/16 to 5/16 or
11/16 or
21/16 to 24/16
"0" level for 25/16 or more
Missing mid-bit-cell transition

NRZI - A transition that is 1/4 bit time or more out of phase. More than two transitions in half bit cell

COLLISION DETECTION BY BIT COMPARISON

An additional mechanism of bit comparison is provided. This mechanism compares the "signature" of the transmitted data to the signature of the received data for the duration of the collision window (one slot time). (Wherever 588 configured to internal collision detection.)

A collision is reported if, after the transmission of the Opening Flag, any of the following conditions becomes true:

1. Half a slot-time has elapsed and Carrier Sense did not go active.
2. Half a slot-time + 16 bit times have elapsed and an opening flag was not yet recognized (16 bit times are a margin factor).
3. Carrier Sense went inactive after an opening flag was received - Transmit still active.
4. Collision window has elapsed and Transmit signature differs from Receive signature.

In mode 1, internal collision detect works only with transceivers that do not return the transmitted signal. In that case, detection of a carrier during transmission indicates a collision.

NOTE:
For broadband applications the slot time is usually twice the round trip delay.

5.0 82588 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The 82588 includes a set of features for improving reliability and testability.

The 82588 offers four diagnostic services: first, monitoring the transmission and reception of frames; second, gathering statistics and diagnostics about the network; third, diagnostic support for a particular station on the network; fourth, a means to test the proper operation of the chip itself.

5.1 Transmission/Reception Error Reporting

The 82588 stores status information after completing transmission or reception of every frame. If transmission or reception is successful, the OK status bit is set. If transmission is unsuccessful, the cause is given in the status registers.

The 82588 reports on the following events after each transmitted frame:

— Lost Carrier Sense: Carrier sense signal did not become active or was lost during transmission.

— Lost Clear-to-Send: Clear to send signal was removed during transmission.

— DMA underrun occurred because the system bus did not keep up with the transmission.

— The number of collisions equals the maximum allowed.

Any of these events causes the report of an unsuccessful transmission.

The 82588 checks each incoming frame and reports on the following errors:

— CRC error: incorrect CRC in a well aligned frame.

— Alignment error: incorrect CRC in a misaligned frame. A misaligned frame with a correct CRC is not reported by the 82588. (Considered a good frame.)

— Frame too short: the frame is shorter than the configured value for minimum frame length.

— No EOF flag: valid only in Bitstuffing mode, carrier went inactive before EOF flag detection.

— Overrun: the frame was not completely placed in memory because the system bus did not keep up with coming data.

The occurrence of any of these events causes the report of an unsuccessful reception.

5.2 Network Planning and Maintenance

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82588 provides a rich set of network diagnostics that can serve as the basis for a network management entity. The features include: gathering network activity information, saving all frames that appear on the link (optional), and locating opens or shorts on the link.

Network activity information is provided in the status available to the host after each frame is transmitted. The activity indicators are:

1. Number of collisions: the number of collisions the 82588 experienced in attempting to transmit this frame.
2. Deferred transmission: indicates if the 82588 had to defer to traffic on the link during the first transmission attempt.

The 82588 can be configured in the Promiscuous Mode. In this mode the 82588 captures all frames transmitted on the network regardless of the Destination Address. This mode is useful in implementing a monitoring station to capture all frames for network analysis.

The 82588 is capable of determining if there is a short or open circuit anywhere in the network using the Time Domain Reflectometry (TDR) command. When a TDR command (see section "Controlling the 82588") is issued, the chip transmits a pulse train and measures the reflection return time. If the network is properly terminated, there are no reflections so the timer runs out and the user is notified that there are no link problems. If a problem is detected, the distance to the reflection source and reason (short or open) are reported.

5.3 Station Diagnostics

To support Station Diagnostics, the 82588 can be configured to External Loopback.

In this mode the 82588 operates full duplex at full speed. The actual maximum throughput depends on bit rate and system bus speed. The transmitter to receiver interconnection can be placed anywhere between the 82588 and the link.

5.4 82588 Self Testing

The 82588 provides several features for self testing.

The 82588 can be configured to Internal Loopback. In this mode, the 82588 disconnects itself from the internal or external Serial Interface Units, and any frame transmitted is received immediately. The 82588 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock. The 82588 can be configured in the External loopback mode. In this mode, the Transmitted signal at full data rate comes out at the TxD pin and can be returned to the RxD pin through any external circuitry. This works full duplex at the full data rate. It tests the complete transmitter, receiver and the external path. Equality between the transmitted and received frames implies that a large portion of the chip operates correctly. In addition, internal loopback can be used in conjunction with inhibiting the source address insertion and/or CRC insertion by the chip. For example, in internal loopback, if a frame is transmitted with an erroneous CRC (using CRC inhibition), the CRC checking mechanism must detect a CRC error. In internal loopback the transmission and reception occurs at one fourth the programmed data rate. The Dump Command causes the 82588 to write 63 bytes of its internal registers to memory. This is a very powerful capability that can serve as the basis for comprehensive diagnostics.

There are parts of the chip, in particular the logic, that uses the Backoff random number generator that cannot be checked from the outside. The Diagnose Command initiates a self test procedure that exercises these otherwise unaccessible registers and counters, and reports the result.

6.0 INITIALIZING/CONFIGURING THE 82588

6.1 Initializing the 82588

A hardware or software reset brings the 82588 to its initial state where the 82588 is put into the idle state with the transmitter, receiver clock generator, interrupts, etc., inactive. To use the 82588 it has to be configured with a set of parameters.

6.2 Configuring the 82588

This is done using the CONFIGURE command. This command initiates the writing of the configuration parameters into the 82588 using DMA. The configuration parameters are shown in Figure 10.

	Configurable Parameters	Default Values on Reset
1.	Serial Interface	*
2.	Sampling Rate	*
3.	Oscillator Range	*
4.	FIFO Limit	*
5.	Chaining	*
6.	Buffer Length	0
7.	Preamble Length	8
8.	Address Length	6
9.	No Source Address Insertion	0
10.	Promiscuous Mode	0
11.	Broadcast Disable	0
12.	CRC-16/CRC-32	0
13.	No CRC Insertion	0
14.	Padding	0
15.	Bitstuffing/EOC	0
16.	Min Frame Size	64
17.	Interframe Spacing Time	96
18.	Slot Time	512
19.	Number of Retries	15
20.	Linear Priority	0
21.	Back-off Method	0
22.	Manchester/NRZI	0
23.	Diff. Manch/Manchester	0
24.	CRS Filter	0
25.	Internal CRS	0
26.	CDT Filter	0
27.	CDBBC	0
28.	Internal CDT	0
29.	INT Loopback	0
30.	EXT Loopback	0
31.	Transmit on No CRS	0
32.	Exponential Priority	0

*these parameters must be configured after RESET

Figure 10. Configuration Parameters

BYTE	7	6	5	4	3	2	1	0
0	BYTE COUNT (L.S.B)							
1	BYTE COUNT (M.S.B)							
2	CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT			
3	BUFFER LENGTH							
4	EXT LP.BCK	INT LP.BCK	PREAM LEN		NO SRC ADD INS	ADD LEN		
5	BOF METD	EXP PRIO			DIF.MAN /MAN	LIN PRIO		
6	INTER FRAME SPACING							
7	SLOT TIME (L)							
8	RETRY NUMBER				CDBBC	SLOT TIME (H)		
9	PAD	BIT STUFF	CRC16	NCRC INS	TON NCRS	MAN /NRZ*	BC DIS	PRM
10	CDT SRC	CDTF			CRS SRC	CRSF		
11	MINIMUM FRAME LENGTH							

CONFIG PARAMETER FORMAT

231368-8

Figure 11. Configuration Block

All but 5 parameters have a default value after reset. Figure 11 shows the configuration block in memory on issuing the configure command. The first two bytes contain the number of bytes (byte count) which follow. Note that all the parameters not having a default value on reset are contained in the first bytes after the byte count field. The minimum byte count is 1 and maximum 10 - values greater than 16 are interpreted as module 16. (Meaning only the 4 lsb bits are used). After reset a configure command with byte count of minimum 1 is essential for the operation of the 82588.

6.3 Configuration Parameters

1. Serial Interface Mode - Byte 2, Bit 6

   This bit selects between high integration (mode 0) and high speed (mode 1) modes. It also selects the function of the $\overline{TXC}$/X1, $\overline{RXC}$/X2 and TCLK/CRS pins.

   In High Integration Mode, the 82588 generates its Internal clock from the x1, x2 input; encodes/decodes the data waveform (either Manchester, Differential Manchester or NRZI); generates the Carrier Sense and (optionally) the Collision detect from the incoming RXD signal.

   In High Speed Mode, the Clock Generation, Data Encoding/Decoding, Carrier Sense and Collision Detect, are generated externally.

   0 - mode 0, high integration

   1 - mode 1, high speed

   Default setting: None

2. Sampling Rate - Byte 2, Bit 5

   Valid only in High Integration Mode. It specifies the ratio between the frequency of the sampling (serial) clock and the data bit rate:

   0 - 8×

   1 - 16×

   Default setting: None

3. Oscillator Range - Byte 2, Bit 4

   Valid only in High Integration Mode. It specifies the frequency range of the sampling (serial) clock:

   0 - high range (1 – 16 MHz)

   1 - low range (DC – 1 MHz)

   Default setting: None

4. FIFO - Limit - Byte 2, Bits 0–3

   Specifies the byte count in the 16 byte FIFOs at which the 82588 is to resume requests for transfer of data to/from memory. This parameter is used for the Transmit FIFO - it's 2's complement is used for the Receive FIFO.

   Default setting: None

5. Chaining - Byte 2, Bit 7

   If this bit is set, the 82588 switches the DMA during reception channel whenever the number of bytes transferred to memory equals the Buffer Length parameter.

   0 - Single Buffer reception

   1 - Multiple Buffer reception

   Default setting: None

6. Buffer Length - Byte 3, Bits 0–7

   Specifies the length of the buffer, after which the DMA channel is switched. This parameter is only valid when the Chaining bit is set. It is specified in units of 4 bytes. Buffer length of 0 is interpreted as 1024 bytes.

   Default setting: 0 (= 1024 bytes)

7. Preamble Length - Byte 4, Bits 4–5

   Selects the length of the Preamble (including the SFD).

   00 - 2 bytes

   01 - 4 bytes

   10 - 8 bytes

   11 - 16 bytes

   Default setting: 10 (= 8 bytes)

8. Address Length - Byte 4, Bits 0–2

   Determines the length, in bytes, of the address that the 82588 refers to. This applies to source, destination, Multicast, and broadcast addresses. Address length of 7 is treated as zero.

   Default setting: 6 bytes

9. No Source Address Insertion - Byte 4, Bit 3

   0 - Source address is inserted by the 82588 during transmission

   1 - Source address insertion is disabled (source address is provided by the user in the transmit block)

   Default setting: 0

10. Promiscuous Mode - Byte 9, Bit 0

    When set, causes 82588 to receive all frames regardless of their destination address.

    0 - Promiscuous mode off

    1 - Promiscuous mode on

    Default setting: 0

11. Broadcast Disable - Byte 9, Bit 1

    Disables reception of any frame with a broadcast destination address. Protects against overloading stations with limited resources. Note that when promiscuous mode (when set) is on, Broadcast is always enabled.

    0 - Broadcast enabled

    1 - Broadcast disabled

    Default setting: 0

12. CRC-16/CRC-32 - Byte 9 Bit 5

    0 - 32 bit Autodin - 11 CRC (IEEE 802.3)

    1 - 16 bit CCITT CRC (HDLC)

    Default setting: 0

13. No CRC Insertion - Byte 9, Bit 4

    0 - CRC is inserted by the 82588 after the information field

    1 - No CRC insertion (CRC is provided by the user)

    Default setting: 0

14. Padding - Byte 9, Bit 7

    Valid for bitstuffing only. If padding is set, the frames shorter than 1 slot time will be padded with flags to the shortest frame that is longer than 1 slot time.

    0 - Padding off

    1 - Padding on

    Default setting: 0

15. Bitstuffing/EOC - Byte 9, Bit 6

    0 - End of Carrier Framing (i.e., IEEE 802.3)

    1 - Bitstuffing Framing (HDLC, with 01111110 flag and zero bit insertion/deletion)

    Default setting: 0

16. Min-Frame-Size - Byte 11, Bit 0–7

    The minimum frame size, (not including preamble) in bytes. The 82588 sets a status bit if a shorter frame is received. Note that frames shorter than 6 bytes are discarded with no status update.

    Default setting: 64 bytes

17. Interframe Spacing - Byte 6, Bits 0–7

    Specifies the time period, in bit times, that the 82588 must wait after detecting that the Carrier Sense dropped and before it can begin transmission or reception of a frame. The minimum value is 12 and any value less than that will default to 12.

    Default setting: 96

18. Slot Time - Byte 7, Bits 0–7; Byte 8, Bits 0–2

    The slot time for the network, in Bit Times. This value is used in calculating backoff and Linear Priority delays. It must be longer than the maximum roundtrip time of a frame in the network plus the jam time. Zero is interpreted as 2**11.

    Default setting: 512

19. Number of Retries - Byte 8, Bit 4–7

    The maximum number of retries, due to collisions, that the 82588 performs. This information gives the CPU the choice of aborting transmission after a programmed number of collisions.

    Default setting: 15

20. Linear Priority - Byte 5, Bit 0–2

    The number of slot times that the 82588 waits after Inter Frame Spacing or after Backoff, before enabling transmission. A higher number reduces the priority.

    Default setting: 0

21. Backoff Method - Byte 5, Bit 7

    Determines the method for starting the backoff timer:

    0 - According to the IEEE 802.3 standard; immediately
    after the end of transmission

    1 - After the interframe spacing period expires

    Default setting: 0

22. Manchester/NRZ(I) - Byte 9, Bit 2

    This bit specifies the data encoding/decoding method - it is mode dependent:

	High Integration	High Speed
0 -	NRZI (*)	NRZ
1 -	Manchester	Manchester (**)

    * Bitstuffing should also be applied

    ** Manchester encoding is performed on transmitted data (TXD)

    NRZ data should, however, be provided for reception (External Manchester decoding)

23. Differential Manchester/Manchester - Byte 5, Bit 3

    Valid only in High Integration Mode provided that Manchester is set in preceding parameter. This bit specifies the data encoding/decoding method:

    0 - Manchester

    1 - Differential Manchester

    Default setting: 0

24. CRS - Filter - Byte 10, Bits 0–2

    Specifies the required width of CRS in bit times, before it is recognized as being active. Carrier sense deactivation is recognized immediately.

    Default setting: 0

25. CRS Source - Byte 10, Bit 3

    Valid only in High Speed Mode. Specifies whether Carrier Sense is to be performed internally or externally (via CRS pin).

    0 - External

    1 - Internal

    Default setting: 0

26. CDT-Filter - Byte 10, Bit 4–6

    Specifies the required width of CDT, in bit times, before it is recognized that a collision has occurred.

    Default setting: 0

27. CDT Source - Byte 10, Bit 7

    Specifies for both High Integration and High Speed Modes whether Collision Detect is to be performed internally or externally (via CDT pin):

    0 - External

    1 - Internal

    Default setting: 0

28. CDBBC - Byte 8, Bit 3

    Valid only in High Integration Mode. Specifies whether Collision Detect by Bit Comparison is to be performed.

    0 - CDBBC off

    1 - CDBBC on

    Default setting: 0

29. INT - Internal Loopback - Byte 4, Bit 6

    When set, the 82588 disconnects itself from the serial link and logically connects TXD to RXD and TXC to RXC.

    Note: If set, it overrides EXT-loopback

    0 - Internal Loopback off

    1 - Internal Loopback on

    Default setting: 0

30. EXT - External Loopback - Byte 4, Bit 7

    When set, the 82588 will transmit and receive simultaneously, at full rate. This allows checking of external hardware as well as the serial link to the transceiver interface. The user is responsible for external transmit-receive interconnection. It is overridden by INT-Loopback.

    0 - External Loopback off

    1 - External Loopback on

    Default setting: 0

31. Transmit on No CRS - Byte 9, Bit 3

    When set, allows transmission even if there is no CRS back from the transceiver. Required for transceivers that do not return carrier during transmission.

    0 - Transmit only when CRS present

    1 - Transmit only when no CRS present

    Default setting: 0

32. Exponential Priority - Byte 5, Bit 4-0

    Extends the range from which the random number for back-off is selected (i.e., first collision is biased to a number different than 0, 1)

    Default setting: 0

7.0 CONTROLLING THE 82588

This chapter specifies how to control the 82588, from the user's point of view. It starts with the definition of the command and status registers. It then specifies the behavior of the 82588 during execution of operations and reception of frames. The next chapter provides the detailed format and meaning of all the operations and their status.

The 82588 performs OPERATIONS, such as transmitting a frame, enabling reception, configuring the chip, performing diagnostics, or aborting an operation. An operation is initiated by the CPU, by writing a COMMAND into the command register in the 82588. There are three groups of operations: EXECUTION, RECEPTION, and STATUS POINTERS CONTROL (in addition to a Software Reset). The first two groups correspond to two logical units of the 82588: the EXECUTION UNIT and the RECEIVE UNIT. The third group provides control for the status registers.

There is a set of EVENTS that may occur asynchronously in response to commands (typically as a result of the behavior of the link). These events include: end of incoming frame, frame aborted, execution of a command is completed, execution aborted. The 82588 reports on these events by updating a set of status registers and raising the INTERRUPT signal. There are four status registers (STATUS 0 → STATUS 3), which are all read by the CPU, from the same address. An internal two bit POINTER is used to determine which register is being read. The pointer is advanced to the next status register automatically when status is read. It can also be set explicitly by a field in the command. Status pointer progress may be disabled by a command that fixes it to a particular status register and allows reading of the same register. Information read from any of the first 3 status registers is valid provided Interrupt signal is high or Bit 7 of Status 0 is set. Information of status register 3 is continuously updated, provided the register pointer is not fixed at three.

Status is updated and interrupt raised following an event only when the interrupt signal is low. When the interrupt is high, the CPU can read the status. The interrupt is cleared by a command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set. Note that it is the responsibility of the CPU to clear interrupt in order to prevent a dead lock.

The 82588 interacts with the DMA controller via two REQUEST/ACKNOWLEDGE lines (referred to as two CHANNELS). The 82588 requests the transfer of bytes, and the DMA performs and acknowledges the transfer. Frames and parameters are transferred via the two channels from/to memory. Commands allocate a specific channel for each operation.

The 82588 can be configured to support receive buffer CHAINING. In this mode, two channels must be allocated to the reception of frames and the 82588 switches channels when a buffer is full. It issues a "Request Alternate Buffer" interrupt, in order to allow the CPU to set up the alternate buffer on the other channel.

7.1 The Command Register

This section specifies the format of the 82588 command set. A command is given to the 82588 by writing into the command register. The command can be issued at any time, but in case it is not accepted, the operation is treated like a NOP and will be ignored (although the INT and the PTR will be updated).

Format:

7	6	5	4	3	2	1	0
INT. ACK.	PNTR		CHNL	OPERATION			

COMMAND FORMAT

231368-9

7.1.1 OPERATIONS (BITS 0–3)

The OPERATION field initiates a specific operation. There are three groups of operations that can be initiated independently: EXECUTION, RECEPTION, or STATUS POINTER CONTROL.

The 82588 implements the following EXECUTION operations (shown with their codes):

- * NOP — 0
- * IA-SETUP — 1
- * CONFIGURE — 2
- * MC-SETUP — 3
- * TRANSMIT — 4
- * TDR — 5
- * DUMP — 6
- * DIAGNOSE — 7
- * RETRANSMIT — 12
- * ABORT — 13

Execution operations except ABORT are accepted only when the Execution unit is IDLE. ABORT is only accepted if the Execution unit is active.

Following are the reception operations (and their codes):

- * RCV-ENABLE - 8
- * ASSIGN ALT BUF (chaining only) - 9
- * RCV-DISABLE - 10
- * STOP-RCV — 11

RCV-ENABLE is accepted only when the Receive Unit is IDLE. The other receive operations are only accepted if the Receive Unit is NOT IDLE.

Status Pointer Control Operation:

- * FIX PTR — 15 (1)
- * RLS PTR — 15 (0)

The bit in brackets is Bit #4 of the command (channel bit).

Format:

	7	6	5	4	3	2	1	0
STATUS 0	INT.	RCV	EXEC	CHNL	EVENT			
STATUS 1	RESULT 1							
STATUS 2	RESULT 2							
STATUS 3	RCV CHNL	RCV STATE		BUFF CHNG NO. OF BUF.		EXEC CHNL	EX STATE	

231368–10

The last operation is software reset:

- * RESET — 14

RESET can always be issued. It has the same effect as a hardware reset.

Operations that are not accepted are simply ignored, (although INT and PTR are updated).

7.1.2 CHANNEL ALLOCATION (BIT 4)

The CH field selects the channel for the operation that is requested. Note this field applies only for IA-SETUP, CONFIGURE, MC-SETUP, (RE) TRANSMIT, DUMP and RCV-ENABLE. If the selected channel is already allocated the operation is ignored (acts like NOP).

7.1.3 SETTING THE POINTER

The PTR field specifies the new value of the internal pointer. The pointer selects the next status register to be read. It is advanced (module 4) automatically when status is read. Automatic advancement may be disabled by issuing the FIX PTR command.

7.1.4 ACKNOWLEDGING INTERRUPT (BIT 7)

The INT-ACK bit, if set, causes the interrupt hardware signal and the interrupt bit to be cleared. This is the only way to clear the interrupt bit and reset the interrupt signal other than by a reset.

7.2 The Status Registers

There are four Status Registers that may be read by the user. They are numbered 0–3. The register that the internal pointer points to, is read by doing a "read" from the 82588. STATUS 1 and STATUS 2 are also called "result Registers."

The 82588 provides the information about the last operation that was executed, or the last frame received, via the first three status registers. The fourth status register provides the state of the chip itself.

STATUS 0, STATUS 1, and STATUS 2 are only updated when INT = 0. STATUS 3 is updated whenever an update is needed. Status register 3 is **not** updated if the pointer is pointing to it.

7.2.1 STATUS 0

The first status register (STATUS 0) includes the interrupt bit and the cause of the interrupt. Information about the relevant channel is also provided.

7.2.1.1 Event (Bits 0–3)

The event field specifies the reason why the 82588 needs attention of the CPU.

The following events may occur (shown with their codes):

* IA-SET-DONE - 1
* CONFIGURE-DONE - 2
* MC-SETUP-DONE - 3
* TRANSMIT-DONE - 4
* TDR-DONE - 5
* DUMP-DONE - 6
* DIAGNOSE-PASSED - 7
* END OF FRAME - 8
* REQUEST ALT BUFFER - 9 (chaining only)
* RECEPTION ABORTED - 10
* RETRANSMIT-DONE - 12
* EXECUTION-ABORTED - 13
* DIAGNOSE-FAILED - 15

7.2.1.2 Channel (Bit 4)

The CH bit indicates which channel was allocated for the operation that caused the event. Valid for all events except for Request Alternate Buffer.

7.2.1.3 Execution (Bit 5)

The EX bit indicates the completion of an execution operation. Event number 13 indicates the abortion of the operation.

7.2.1.4 Receive (Bit 6)

The RCV bit indicates that the reception of a frame was completed, or a new buffer is required to allow reception to proceed. Event numbers 8–10 provide the details.

7.2.1.5 Interrupt (Bit 7)

The INT bit is set together with the hardware interrupt signal. Setting the INT bit indicates the occurrence of an event. This bit is cleared by any command whose INT-ACK bit is set.

7.2.2 STATUS 1/STATUS 2 (RESULT REGISTERS)

These registers hold the result of completing an operation. STATUS 1 is the least significant byte and STATUS 2 is the most significant. The Result Registers are only updated when any of the following occur:

— TRANSMIT-DONE

— TDR-DONE

— RETRANSMIT-DONE

— END OF FRAME

7.2.3 STATUS 3

The last status register holds the state of the 82588: three lower bits for the Execution unit, five higher bits for the Receiver unit.

7.2.3.1 Execution State (Bits 0–1)

The EX-CH field specifies the state of the Execution Unit. It can take one of the following values:

0 - IDLE:	The Exec Unit is disabled
2 - ACTIVE:	The Exec Unit is actively executing an operation
3 - ABORT IN PROG:	The execution of an operation was aborted, but the completion event was not yet issued.

7.2.3.2 Exec Channel (Bit 2)

The EX-CH bit indicates which channel is allocated for transferring parameters from/to memory. It is only valid when the Execution Unit is active, performing a parametric execution.

7.2.3.3 Buffer Chaining - Number of Buffers (Bit 3–4)

Specifies the number of available buffers for reception, when configured to "Buffer Chaining." It can take one of the following values:

0 - no buffers are available
1 - one buffer is available (automatic on RCV-enable)
2 - two buffers are available

7.2.3.4 Receive State (Bits 5–6)

The RCV-STATE field specifies the state of the Receive Unit. It can take one of the following values:

0 —	IDLE	The Receive Unit is disabled
1 —	READY	The Receive Unit is set up for an incoming frame
2 —	ACTIVE	The Receive Unit is actively transferring bytes to memory (provided buffer is available)
3 —	STOP IN PROGRESS	The Receive Unit is active and will be disabled following the reception of current frame

7.2.3.5 Receive Channel (Bit 7)

The RCV-CH bit indicates which channel is allocated for transferring incoming data to system memory. It is valid only when the Receive Unit is **NOT IDLE**. When configured for Buffer Chaining, at least one buffer should be available. 82588 assumes that one buffer is available whenever the receiver is enabled. The DMA controller must be initialized and ready to transfer data prior to enabling the receiver.

7.3 Performing Execution Operations

The 82588 has a repertoire of 11 execution operations. These operations perform transmission, configuration, diagnostics and abortion. All the operations are initiated by writing into the Command Register. This causes the Execution unit to become Active. Some of the operations read parameters from memory.

The parameters (for IA-SETUP, CONFIGURE, MC-SETUP, TRANSMIT and RETRANSMIT) are organized in a block that starts with a 16 bit byte-count. The byte count specifies the length of the rest of the block. Before beginning the operation, the DMA pointer of the selected channel must point to the first byte of the byte count. There is no restriction on the memory structure of the frame as long as for every DMA request from the 82588 it receives the next byte of the frame. Transferring the bytes is the job of the DMA controller or the CPU. If the DMA does not keep up with transmission (i.e., causes an underrun), the transmission is terminated and an underrun is indicated.

The 82588 requests the 2 bytes of byte-count on the allocated channel and determines the length of the parameter block. It then requests the parameters and starts the execution of the operation.

Upon completion of the operation, (when interrupt is LOW) the Exec unit does the following: it updates STATUS 0; updates STATUS 1 and STATUS 2 (for TRANSMIT, TDR or RETRANSMIT); raises the interrupt signal; and goes IDLE.

The NOP operation is ignored, exactly like any operation that is initiated during the execution of another one.

The ABORT operation causes a request for abortion of the operation that is in progress. In some cases the operation cannot be aborted anymore (e.g., for a Transmit operation when the whole frame was already transmitted). In these cases the Abort operation is ignored. Non-parametric executions like: TDR and DIAGNOSE cannot be aborted.

The DUMP operation causes a set of internal registers to be written to memory over the allocated channel. It is completed by updating STATUS 0 and raising the interrupt.

7.4 Reception of Frames

The 82588 receives and passes to memory all frames whose address matches the individual, multicast or broadcast address. Reception of frames is pipelined, i.e., the reception of a frame can start before the end of frame processing of the previous frame is done. 82588 is configured to either "chaining" or "consecutive" mode. In consecutive mode the whole incoming frame is passed to a single buffer in memory via the allocated channel. In chaining mode, the frame is received and deposited in a series of fixed size buffers using the two available DMA channels alternately. The chained buffer size if configurable in steps of 4 bytes to a maximum of 1K bytes.

7.4.1 CONSECUTIVE MODE (SINGLE BUFFER MODE)

Before the reception starts, the DMA pointer of the respective channel points to the first byte of the available structure of the frame. Transferring the bytes is the job of DMA controller or the CPU.

If the receiver is ready and a frame arrives, the 82588 requests the transfer of bytes to memory, using DMA. After transferring all the bytes (addresses, control and information) the 82588 transfers two "frame status" bytes to memory. Upon completion of the reception, and when interrupt is LOW, the Receive Unit does the following: it updates STATUS 0 with an "end of Frame" event; loads the frame byte count, which includes the frame status bytes, into STATUS 1 and STATUS 2; raises the interrupt signal; and goes to READY state. (The DMA Controller points to the byte following the frame status.)

If the receiver is not ready when the first byte of the frame arrives, then the frame is not received. No status is updated. Disabling reception after the first byte has passed to memory, causes the rest of the frame to be ignored. An INTERRUPT with "Receive Aborted" event is issued.

Reception can begin even when the interrupt that indicates the completion of execution or reception is still pending. This capability allows pipelining of the transfer to memory of a frame by the 82588 and the DMA controller, with the CPU handling the completion of the previous execution or reception (including the interrupt latency). The entire frame gets deposited in a single memory buffer of 64K bytes maximum size.

7.4.2 CHAINING MODE (MULTIPLE BUFFER MODE)

Before the reception starts, the DMA pointer of the first DMA channel points to the first byte of the available buffer. If receiver is ready (one buffer allocated) and a frame arrives, the 82588 generates an interrupt with a "Request Alternate Buffer" event that notifies the CPU to allocate an additional buffer. Simultaneously, the 82588 requests the transfer of bytes to memory. Note that if an additional buffer is available when the first byte arrives, then the "Request Alternate Buffer" event is not issued. This may happen when more than one frame is received sequentially.

When the currently used buffer is full and an additional buffer is available the 82588 makes it the current one (i.e., switches buffers) and continues transferring bytes to memory using the other DMA channel. If an additional buffer is not available, the receiver deasserts the DMA Request signal and suspends further data transfers to memory. The CPU should then provide another buffer or disable receive. If a new buffer is provided, byte transfers to memory continues. If a Disable Receive command is issued then the rest of the frame is ignored.

This process of switching from channel to channel continues until the end of the frame arrives. At this point, two "frame status" bytes are transferred to memory; buffer switch occurs; STATUS 0 is updated; the byte count of the frame is loaded into STATUS 1 and STATUS 2; if a new buffer is available, then the 82588 is ready to receive the next frame. Otherwise, reception is suspended. In both cases, and End of Frame event is issued.

If the receiver is not ready, when the first byte of the frame arrives, then the whole frame is ignored - status is not updated. Disabling reception after the first byte was passed to memory causes the rest of the frame to be ignored, and an interrupt with "Receive Aborted" event to be issued.

If reception is suspended (Ready, but with no buffers assigned) when the first byte of the frame arrives, the receiver interrupts the CPU for a buffer. The user may provide a buffer or disable Reception.

8.0 OPERATIONS AND STATUS

This section specifies all the operations that the 82588 can execute and the events it can generate. The semantics, as well as the format of the parameters, status, and result, are specified. The 4 bit op-code is shown in parentheses.

The value of byte count (where applicable) does not include the 2 bytes of the field itself. All shaded areas mean they are "reserved."

8.1 Operations

8.1.1 NOP (00)

This operations does not affect the 82588. It has no parameters and no result.

8.1.2 IA-SETUP (01)

This operation sets up the individual address of the 82588. This address is inserted as the source address during transmission and used for address filtering during reception. (Both can be disabled via configuration parameters.) The Byte Count should match the "Address Length" configuration parameter. The length of the address (in bytes) that is set up is determined by the minimum of:

"Address Length" configuration parameter or the value of the Byte Count field.

Only the 4 least significant bits of the byte count are used.

NOTE:
The first bit of the first byte must be a zero.

The format of the parameter is as follows:

7	6	5	4	3	2	1	0
BYTE COUNT (L.S.B.)							
BYTE COUNT (M.S.B.)							
INDIVIDUAL ADDRESS LSB BYTE							0
INDIVIDUAL ADDRESS MSB BYTE							

231368-11

An interrupt with the event IA-SETUP-DONE is issued when this operation is done (unless ABORT was executed in the meantime).

8.1.3 CONFIGURE (02)

This operation configures the 82588. The length of the configurations block that is modified is determined by the byte count. If the byte count is less than 10, then only part of the parameters are updated. Only the least significant 4 bits of byte count are used.

Note that 3 bits (Mode, SMPLG RATE, and OSC RANGE) have an effect only during the first Configure command after RESET.

The format of the parameters is as follows:

7	6	5	4	3	2	1	0
BYTE COUNT (L.S.B)							
BYTE COUNT (M.S.B)							
CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT			
BUFFER LENGTH							
EXT LP.BCK	INT LP.BCK	PREAM LEN		NO SRC ADD INS	ADD LEN		
BOF METD	EXP PRIO			DIF.MAN /MAN*	LIN PRIO		
INTER FRAME SPACING							
SLOT TIME (L)							
RETRY NUMBER				CDBBC	SLOT TIME (H)		
PAD	BIT STUFF	CRC16	NCRC INS	TON NCRS	MAN /NRZ*	BC DIS	PRM
CDT SRC	CDTF			CRS SRC	CRSF		
MINIMUM FRAME LENGTH							

231368-12

The interpretation of the fields is as follows:

Byte 0	FIFO Limit (Bits 0–3) OSC RANGE (Bit 4) SMPLG RATE (Bit 5) MODE (Bit 6) CHAINING (Bit 7)	FIFO limit High or low frequency range of oscillator Sampling rate. 8× or 16× external oscillator High integration or high speed mode Receive buffer chaining
Byte 1	BUFFER LENGTH	Buffer length used for chaining
Byte 2	ADR LEN (Bits 0–2) NO SRC ADDR INS (Bit 3) PREAM LEN (Bits 4–5) INT LP BCK (Bit 6) EXT LP BCK (Bit)	Address length Address control location Preamble length Internal Loopback External Loopback
Byte 3	LIN PRIO (Bits 0–2) DIF MAN/MAN (Bit 3) EXP PRIO (Bits 4–6) BOF METD (Bit 7)	Linear priority Differential/Manchester Encoding Exponential Priority Exponential backoff method
Byte 4	INTER FRAME SPACING	Inter frame spacing
Byte 5	SLOT TM (L)	Slot time, low byte
Byte 6	SLOT TM (H) (Bits 0–2) CDBBC (Bit 3) RETRY NUM (Bits 4–7)	Slot time, high part Collision Detection by bit comparison Number of transmission retries on collision
Byte 7	PRM (Bit 0) BC DIS (Bit 1) MANCH/NRZ (Bit 2) TON NCRS (Bit 3) NCRC INS (Bit 4) CRC-16/CRC-32 (Bit 5) BIT STF (Bit 6) PAD (Bit 7)	Promiscuous mode Broadcast disable Manchester/NRZ or Manchester/NRZI encoding Transmit on NO CRS No CRC insertion CRC type Bit stuffing Padding
Byte 8	CRSF (Bit 0–2) CRS SRC (Bit 3) CDTF (Bits 4–6) CDT SRC (Bit 7)	Carrier sense filter bits Carrier sense source Collision detect filter bits Collision detect source
Byte 9	MIN FRAME LEN	Minimum frame length

A Configure-Done interrupt is issued when the operation is done (unless ABORT was executed in the meantime).

8.1.4 MC-SETUP (03)

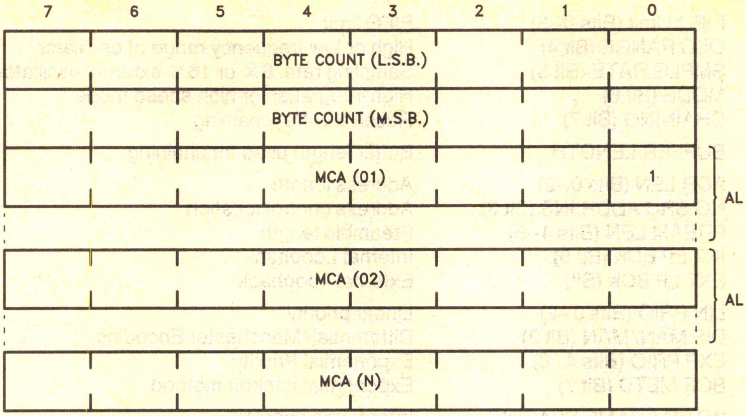
This operation clears and sets up a new 64 bit Multicast Address table. The parameter block may include a number of MC-addresses. Each contributes one bit to the Hash Table.

The first byte of the n-th MC-address in the parameter block begins at byte number n × AL + 1, where AL is the "Address Length" configuration parameter and n is a integer counting from zero.

NOTE:
The least significant bit in the first byte of EACH MC address must be a ONE.

The number of MC addresses that are set up during the operation is equal to Byte Count divided by AL, rounded down, i.e., if the last MC address is incomplete, it is ignored.

The format:



231368–13

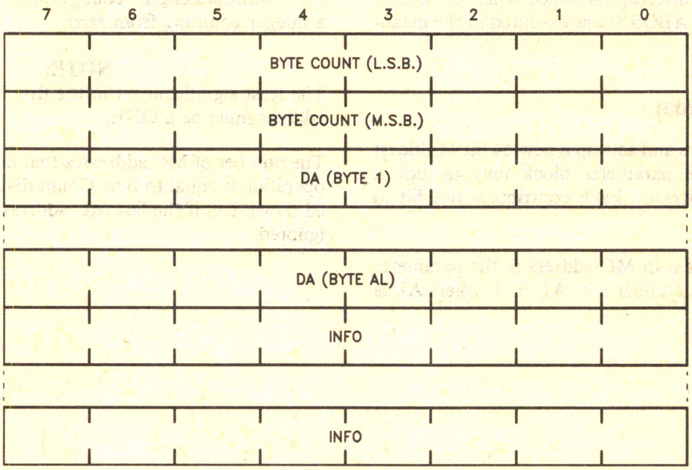
The MC-Setup-Done interrupt is issued after completing this operation (unless ABORT was executed in the meantime).

8.1.5 TRANSMIT (04)

This operation transmits one frame. The parameter block includes the following parameters:

- Destination Address— length determined by the "Address Length" configuration parameter.
- Source Address— **ONLY IF** the no source address insertion configuration parameter is one. The length is determined by "Address Length".
- Information— The length is determined by the Byte Count minus Destination Address Minus Source Address (if any) minus CRC (if any).
- CRC— **ONLY IF** the "No-CRC-insertion" configuration parameter is one. The length is determined by the "CRC-16/32" parameter.

To summarize, the typical transmit operation parameter block includes the Destination Address, and Information fields:



231368–14

The transmit operation will either complete the execution or be aborted by a specific ABORT operation. A Transmit-Done or "Execution Aborted" interrupt is issued upon completion of this operation.

8.1.6 TDR (05)

This operation activates the Time Domain Reflectometer, which is a mechanism to detect open or short circuits (and their distance from the diagnosing station) on the link. There are no parameters. A TDR-Done interrupt is issued upon completion.

8.1.7 DUMP (06)

This operation causes dumping of a set of internal registers. There are no parameters. The 63 registers are dumped to memory on the assigned channel.

The DUMP operation will either complete the execution or be aborted by a specific ABORT operation. A DUMP- DONE or "Execution Aborted" interrupt is issued upon completion of this operation.

8.1.8 DIAGNOSE (07)

The Diagnose Command checks the 82588 timers hardware which includes:

* Exponential Backoff Random Number Generator.
* Exponential Backoff Timeout Counter.
* Slot Time Period Counter.
* Collision Number Counter.
* Exponential Backoff Shift Register.
* Exponential Backoff Mask Logic.
* Time Trigger Logic.

8.1.9 RETRANSMIT (12)

This operation is almost identical to the TRANSMIT operation. The only difference is the RETRANSMIT does not reset the retry counter and the exponential back-off mechanism.

The Retransmit operation will either complete the Execution or be aborted by a specific ABORT operation. A RETRANSMIT-DONE or "Execution Aborted" is issued upon completion of this operation.

8.1.10 ABORT (13)

This operation causes the attempt to abort the completion of an operation under execution. It is valid for: IA-SETUP, CONFIGURE, MC-SETUP, (RE)TRANSMIT and DUMP. It is ignored for any of the above, if the transfer of parameters was completed.

8.1.11 RCV-ENABLE (8)

This operation enables the reception. It is ignored if the receive state is already Ready or Active.

The 82588 receives only frames that pass the address filtering. The frame and its status are placed in memory (by the DMA Controller or CPU) and the byte count of the frame is placed in the Result Registers. The completion of frame reception causes an "End of Frame" event.

8.1.12 ASSIGN ALTERNATE BUFFER (9)

This operation assigns an alternate buffer if reception is ready (and the chip is configured to chaining mode). This operation may be issued when an execution is active and the Ex channel is used for transfer of parameters. In this case ASSIGN ALT BUF will cause the execution to be aborted (the same as ABORT operation). If the on-going execution is thus aborted, an "execution-aborted" interrupt occurs after the reception is complete. An executing command so aborted can then be restarted.

This operation has no parameters and no results. It is usually issued by the CPU following a "Request Alternate Buffer" event.

8.1.13 RCV-DISABLE (10)

This operation causes the reception to be disabled. If transfer of a frame to memory has already begun, then it causes a Reception Aborted event. It has no parameters and no results.

8.1.14 STOP-RCV (11)

This operation requests the disabling of reception. If reception is actually in progress, then it will continue until the end of the frame. If the reception is not in progress, then this command acts like the RCV-DISABLE command.

8.1.15 RESET (14)

This command resets the chip. It has the same effect as hardware reset. Note that the reset command does not reset bits of status registers 0, 1 and 2 other than the Int. field.

8.1.16 FIX PTR (1, 15)

This operation stops the cyclic progress of the read pointer of the STATUS Registers. The pointer is assigned to the register specified in the Pointer Field of the Command.

The channel bit must be set to "1". After this command every read from the status port accesses only the selected (PTR) register. Status 3 register updates stop when it is selected.

8.1.17 RLS PRT (0, 15)

This operation releases the assignment of the read pointer to a specific status register and restores the cyclic progress mode (any read advances the pointer to the next register).

The channel bit must be set to "0". This command with pointer = 0 is executed automatically on reset.

8.2 Illegal Commands

8.2.1 ILLEGAL EXECUTIONS

8.2.1.1 Non-parametric Executions (TDR, DIAGNOSE, ABORT)

TDR, Diagnose will be rejected if the Execution Machine is not idle.

Abort is rejected if issued when Execution Machine is not active, or a non-parametric execution is performed, or transfer of parameters has already been accomplished.

8.2.1.2 Parametric Executions

Will be rejected if one of the following exists:

1. Execution Machine is not idle.
2. The Exec Channel equals the RCV Channel and the RCV Machine is not idle, and it is either configured for Non-chaining Mode or it has at least one channel.
3. The RCV Machine is configured for chaining and it is either active or has been assigned both channels.

8.2.2 ILLEGAL RCV COMMANDS

8.2.2.1 Receive Enable

Will be rejected if one of the following exists:

1. Receive Machine is not idle.
2. The RCV Channel equals the Exec Channel and the Exec Machine already performs a parametric execution.

8.2.2.2 Assign Alternate Buffer

Will be rejected if one of the following exists:

1. The RCV Machine is not configured for Buffer Chaining.
2. The RCV Machine is in IDLE state.
3. An alternate buffer is already assigned.

8.2.2.3 Receive Disable & Stop Receive

Will be rejected if the RCV Machine is in IDLE state.

8.3 Event Statuses

This section specifies all statuses that indicate the occurrence of events This status is always accompanied by an interrupt bit.

8.3.1 IA - SETUP-DONE (01)

This event indicates the completion of an IA-SETUP operation.

8.3.2 CONFIGURE-DONE (02)

This event indicates the completion of a CONFIGURE operation.

8.3.3 MC-SETUP-DONE (03)

This event indicates the completion of an MC-SETUP operation.

8.3.4 TRANSMIT-DONE (04)

This event indicates the completion of the TRANSMIT operation.

The two Result Registers provide information about errors and other diagnostic information. The format of the Result Registers is as follows:

	7	6	5	4	3	2	1	0
RESULT1	TX DEF	HRT BEAT	MAX COLL		NUM OF COLLISIONS			
RESULT2	RCNT COLL		TX OK			LOST CRS	LOST CTS	UNDER RUN

TRANSMIT RESULTS FORMAT

231368-15

Where:

NUMBER OF COLL Number of collisions the frame has experienced (modulo 16).

MAX COLL Transmission failed due to a collision. The configured number of retries is exhausted.

HRT BEAT Collision detect test passed after the previous frame.

TX DEF Transmit deferred due to link activity.

UNDERRUN (*) Indicates that the DMA did not keep up with transmission data rate.

LOST CTS (*) Clear-to-Send lost during transmission.

LOST CRS (*) Carrier Sense lost during transmission, or not set until end of preamble.

TX OK Transmission executed okay.

COLL (*) The transmission of the last frame experienced a collision.

"Number of Collisions" provides redundant information. The setting of one of the bits marked (*) will cause TX-OK bit to be reset.

8.3.5 TDR-DONE (05)

This event indicates the completion of a TDR operation.

The Result Registers indicate whether there is a problem, which type and where.

	7	6	5	4	3	2	1	0
RESULT1	TDR (L)							
RESULT2	OK	XCVR	OPEN	SHORT		TDR (H)		

TDR RESULTS FORMAT

231368-16

Where:

TDR : Twice the radial distance between the 82588 and the location of the problem (in bit times).

SHORT : Indicates there is a short circuit on the transmission line. (Carrier Sense Signal dropped)

OPEN : Indicates the transmission line is not properly terminated. (Collision Detect went active)

XCVR : Indicates a transceiver problem. (Carrier Sense was inactive for 2048 bit time period)

OK : Indicates that no problem was found.

8.3.6 DUMP-DONE (06)

This event indicates that the DUMP operation is completed. The result registers are not updated. Contents of 63 internal registers are transferred to memory on the allocated channel.

The format of the dumped registers is as follows:

7	6	5	4	3	2	1	0	
CHNG	SERIAL MODE	SMPLG RATE	OSC RANGE	FIFO LIMIT				00
BUFFER LENGTH								01
EXT LP.BCK	INT LP.BCK	PREAM LEN		NO SRC ADD INS	ADD LEN			02
BOF METD	EXP PRIO			DIF.MAN /MAN	LIN PRIO			03
INTER FRAME SPACING								04
SLOT TIME (L)								05
RETRY NUMBER				CDBBC	SLOT TIME (H)			06
PAD	BIT STUFF	CRC16	NCRC INS	TON NCRS	MAN /NRZ*	BC DIS	PRM	07
CDT SRC	CDTF			CRS SRC	CRSF			08
MINIMUM FRAME LENGTH								09
1	1	1	1	1	1	1	1	0A
IAR0								0B
IAR1								0C
IAR2								0D
IAR3								0E
IAR4								0F
IAR5								10
TX DEF	HRT BEAT	MAX COLL	0	NUM OF COLLISIONS				11
RCNT COLL	0	TX OK	0	0	LOST CRS	LOST CTS	UNDER RUN	12
TXCRC0								13
TXCRC1								14

231368-17

7	6	5	4	3	2	1	0	
TXCRC 2								15
TXCRC 3								16
RXCRC 0								17
RXCRC 1								18
RXCRC 2								19
RXCRC 3								1A
TMPR 0								1B
TMPR 1								1C
TMPR 2								1D
TMPR 3								1E
TMPR 4								1F
TMPR 5								20
SHRT FRM	NO EOF	1	1	1	1	1	1	21
1	0	RX OK	1	CRC ERR	ALN ERR	0	OVER RUN	22
HASHR 0								23
HASHR 1								24
HASHR 2								25
HASHR 3								26
HASHR 4								27
HASHR 5								28
HASHR 6								29

231368-18

7	6	5	4	3	2	1	0	
HASHR 7								2A
								2B
O.K.	XCVR	OPEN	SHORT					2C
								2D
1	1	1	1	1	1			2E
STTR 0								2F
STTR 1								30
STTR 2								31
STTR 3								32
BUFF.CNT.(L)								33
BUFF.CNT.(H)								34
RCV BYTE CNT.(L)								35
RCV BYTE CNT.(H)								36
0	0	1		1	1	0	1	37
0	1	0		1	0	1	0	38
1	1	0		1	0	0	1	39
EXE BYTE CNT. (L)								3A
EXE BYTE CNT. (H)								3B
EXE DAT. BUF. 0								3C
EXE DAT. BUF .1								3D
EXE DAT. BUF. 2								3E

231368-19

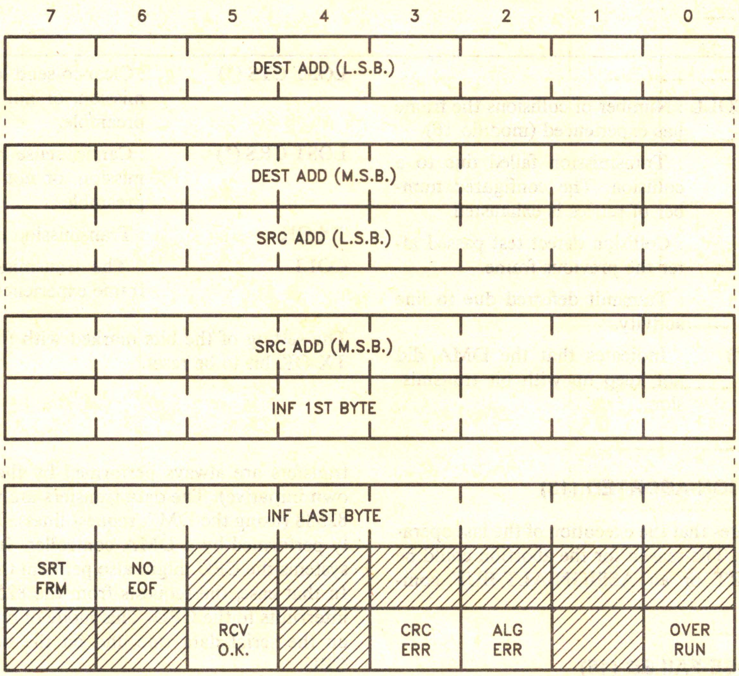
8.3.7 DIAGNOSE-PASSED (07)

This event indicates that the self check procedure was completed successfully.

The format of the received frame is as follows:

8.3.8 END-OF-FRAME (8)

This event indicates the completion of reception of a frame, and depositing of it in memory. The result registers hold the byte count of the frame.



231368-20

Where:

NO EOF	No EOF flag (in bitstuffing only).
SRT FRM	Received frame is shorter than Minimum Frame Length parameter.
OVER RUN	The DMA did not keep up with incoming bits.
ALG ERR	Alignment error.
CRC ERR	CRC error.
RCV OK	Frame received OK.

RCV OK is reset if any of the others is set.

8.3.9 REQUEST ALTERNATE BUFFER (9)

This event is issued in Chaining mode, when reception is enabled, a frame arrives and an additional alternate buffer is not yet assigned.

8.3.10 RECEPTION ABORTED (10)

This event is issued as a result of a RCV-DISABLE operation that causes part of a frame to be discarded.

8.3.11 RETRANSMIT-DONE (12)

This event indicates that the RETRANSMIT operation was completed. It DOES update the result registers. The format of the result registers is identical to the TRANSMIT-DONE, except that the "number of collision" provides essential information.

	7	6	5	4	3	2	1	0
RESULT1	TX DEF	HRT BEAT	MAX COLL		NUM OF COLLISIONS			
RESULT2	COLL		TX OK			LOST CRS	LOST CTS	UNDER RUN

231368-21

Where:

NUMBER OF COLL : Number of collisions the frame has experienced (modulo 16)

MAX COLL : Transmission failed due to a collision. The configured number of retries is exhausted.

HRT BEAT : Collision detect test passed after the previous frame.

TX DEF : Transmit deferred due to line activity.

UNDERRUN (*) : Indicates that the DMA did not keep up with bit transmission.

LOST CTS (*) : Clear-to-send lost during transmission, or not set until end of preamble.

LOST CRS (*) : Carrier sense lost during transmission, or not set until end of preamble.

TX OK : Transmission executed OK.

COLL : The transmission of the last frame experienced a collision.

The setting of the bits marked with (*) will cause the TX-OK bit to be reset.

8.3.12 EXECUTION-ABORTED (13)

This event indicates that the execution of the last operation was aborted. The reason for the abortion could be either an ABORT or an ASSIGN-ALT-BUF command.

8.3.13 DIAGNOSE-FAILED (15)

This event indicates that the self check procedure was completed unsuccessfully.

9.0 SYSTEM INTERFACE

The data bus is 8 bits wide and there is no address bus. The 82588 can interface to wider buses, but only 8 bits will be used. The 82588 is a slave on the bus, i.e., it does not perform memory transfers by itself. The transfers are performed by the CPU or the DMA controller. The 82588 has a very general and simple interface allowing it to operate with a variety of processors and DMA controllers.

There are two kinds of transfers over the bus: Command/Status and Data transfers. The command/status transfers are always performed by the CPU (with its own initiative). The data transfers are requested by the 82588 (using the DMA request lines). They are typically performed by a DMA controller. In relatively slow systems the CPU might also perform the data transfers. In that case, the requests from the 82588 will serve as interrupts to the CPU. This mode of operation depends on the serial data rate and on the dedication of the CPU.

The system interface performs command/status transfers, data transfers, and interrupts.

9.1 Command/Status Transfers

The CPU controls the 82588 by writing into the Command Register and reading from the Status Register. The CPU writes a command by activating the CS input of the 82588, putting the command onto the bus, and activating the WR input of the 82588. In order to read status from the 82588, the CPU activates the CS line and then activates the RD line. The 82588 responds by putting the status onto the bus (during the time that RD is active). There is no address line by which the CPU can select a specific register, within the 82588. The register is selected by the internal 2 bit pointer.

9.2 Data Transfer

Data transfers are controlled by two pairs of REQUEST/ACKNOWLEDGE lines (referred to as two CHANNELS): DMA Request lines (DRQ0 and DRQ1) and DMA Acknowledge lines (DACK0 AND DACK1). Frames and parameters are transferred via the two channels (0 and 1) to or from memory. Commands allocate a specific channel for each operation.

In order to request a transfer from memory, the 82588 activates a DRQ signal on a channel that has been set up for transfer from memory. In response, the DMA controller activates the respective DACK and performs the data transfer. Data is transferred on the bus and written into the 82588 on the rising edge of the WR signal, which is activated by the DMA controller.

In order to request a transfer to memory, the 82588 activates a DRQ signal on a channel that has been previously set up for transfer to memory. In response, the DMA controller activates the respective DACK and performs the data transfer. The 82588 outputs data onto the bus during the time that the RD line is activated by the DMA controller.

9.3 Interrupt

The 82588 reports on completion of an event (see section "Controlling the 82588") by updating a set of status registers and raising the INTERRUPT signal (assuming this signal is initially low). The interrupt is cleared by the command from the CPU where the INTERRUPT ACKNOWLEDGE bit is set. Note that it is the responsibility of the CPU to clear interrupts in order to prevent deadlocks.

9.4 Performance Considerations

This section elaborates on data transfer limitations, bus utilization and interframe spacing considerations. Note, the analysis is only for "steady state," and therefore, should only be regarded as a rough guideline.

9.4.1 TRANSFER RATE LIMITATIONS

The system interface of the 82588 must be able to keep up with the serial link. Assuming f_s is the serial data bit rate, then $f_s/8$ is the serial data rate in bytes/sec. The data rate of the system interface depends on the system clock (f_p) and the number of clocks required to transfer one byte (n).

In order for the interface to keep up with the serial link, the following must hold:

$$\frac{f_p \times 8}{f_s \times n} \text{ is greater than } 1$$

The 82588 uses two on chip 16 byte FIFOs to increase burstiness. A configurable FIFO limit is associated with the FIFOs. The receive FIFO (that is filled by the Receiver Machine) requests the bus only when it is filled above the limit's 2's complement. It transfers bytes in a burst till the FIFO is empty. The transmit FIFO (that is emptied by the Transmit Machine) requests the bus only when it is emptied till the limit. It then transfers a burst of bytes from the system bus till it is full.

Pin Name			Function
$\overline{CS}$*	$\overline{RD}$	$\overline{WR}$	
1 0	x 1	x 1	No transfer to/from Command/Status
0	0	0	Illegal
0	0	1	Read from status register
0	1	0	Write to Command register
$\overline{DACK0}$[$\overline{DACK1}$]*	$\overline{RD}$	$\overline{WR}$	
1 0	X 1	X 1	No DMA transfer
0	0	0	Illegal
0	0	1	Data Read from DMA channel 0 [or 1]
0	1	0	Data Write to DMA channel 0 [or 1]

*Only one of $\overline{CS}$, $\overline{DACK0}$ and $\overline{DACK1}$ may be active at any time.

The 82588 must take into account that it takes a number of clocks to acquire the bus (in the range between N_{Amin} to NA_{max}). In order to prevent underruns (or overruns) the maximum acquisition time must be less than the time to empty the FIFO from the limit.

$$\frac{N_{Amax}}{f_p} \text{ is less than } \frac{LIMIT_E}{f_s/8}$$

i.e.,

$$\text{LIMIT is greater than } \frac{f_s \times N_{Amax}}{f_p \times 8}$$

In order to assure an internal of N_I clocks that the CPU can have the bus between bursts, the following must hold: the time to empty the FIFO till the limit, plus the minimum acquisition time, must be greater than the time between bursts:

$$\frac{16\text{-LIMIT}}{f_s/8} + \frac{N_{Amin}}{f_p} \text{ is greater than } \frac{N_I}{f_p}$$

i.e.,

$$\text{LIMIT is less than } 16 - \frac{f_s \times (N_I - N_{Amin})}{f_p \times 8}$$

9.4.2 BUS UTILIZATION

We define bus utilization as the fraction of time that the bus is not busy servicing the 82588 (including bus acquisition). The following analysis, again, is based on "steady state" and therefore, should be regarded as a guideline only.

The complement of the CPU utilization is the time that the CPU is busy filling the FIFO (t_F), divided by the cycle time from FIFO full till its is full again:

$$CPUUTIL = 1 - \frac{t_F}{\frac{16 - LIMIT}{f_s/8} + t_F + t_A}$$

t_A = acquisition time

To calculate t_F we use the fact that, during one cycle the number of bytes entered into the FIFO equals the number removed:

$$16 - LIMIT + t_A \times f_s/8 + t_F \times f_s/8 = t_F \times f_p/n$$

$$\text{i.e., } t_F = \frac{16 - LIMIT + t_A \times f_s/8}{f_p/n - f_s/8}$$

Therefore, by substituting t_F into CPUUTIL:

$$CPUUTIL = 1 - \frac{f_s \times n}{f_p \times 8}$$

This same expression could be derived directly, as it represents the complement of the ratio of the time to transfer a byte to/from memory, over the time to transfer a byte from/to the link. Note, this expression applies only to the period of transferring, no taking into account the effects between buffers or frames.

9.4.3 BUFFER SIZE CONSIDERATIONS

The 82588 provides support for chaining buffers in memory during reception of frames. The 82588 compares the number of bytes transferred to memory to a programmable buffer limit register. When the count equals the limit and the alternate channel is setup, then the 82588 switches to the alternate DMA channel, resets the counter and issues a "Request Alternate Buffer" interrupt. The CPU interrupt service routine takes care of updating the DMA Controller parameters, issues an "Alternate Buffer Assigned" and acknowledges the interrupt. The minimum buffer size depends on the maximum CPU SERVICE TIME which is composed of the following: the time from issuing the interrupt till the CPU services it, the time the CPU needs to sort out the reason for the interrupt, and the time it takes the CPU to respond. This time depends on the system (not on the 82588), but note, that the data transfer may contend for the bus. Note, activity on the EX channel changes the buffer size considerations.

There are several constraints on the size of the buffer (not taking into account the effects of the case of a partially filled last buffer). The advantage of using short buffers is that, assuming frames of random length, half a buffer will be fragmented (per frame). The smaller the buffer the less memory is wasted. However, the buffer has a minimum size that is determined by the fact that, in order to prevent overruns, the CPU service time must be shorter than the time to fill a buffer.

The advantages of longer buffers is less CPU overhead for buffer switching and memory overhead for storing buffer descriptors. However, the maximum buffer length is 1K.

9.4.4 INTER FRAME SPACING

Inter frame spacing (IFS) is a configuration parameter that is enforced by the link management. The minimum IFS is intended to assure recovery time between completion of transmit or receive to receive, and between transmit to retransmit.

The 82588 supports pipelining between received frames. This feature relaxes the constraints on the IFS. In consecutive mode (i.e., not chaining) the CPU service time, following a frame, must be shorter than the SUM of the IFS and the length of the frame.

Therefore, if the CPU service time is short enough, and the minimum frame long enough, the IFS may be as short as 12 bit times (which is the minimum configurable).

In chaining mode the constraint is more severe, and the IFS time must be longer than the CPU service time. This is due to worst case where the last byte of the frame is placed in the first byte of the buffer.

The 82588 does not perform the retransmission by itself. It only notifies the CPU about a collision, stops transmission and disables transmission for backoff delay, which is in the minimum case equal to IFS. It is desirable (but not mandatory) that the station be able to retransmit within the deferring time (IFS). In order to assure being ready for retransmission, within the deferring time, the IFS must be longer than the CPU service time.

10.0 80188 BASED SYSTEM

Figure 12 shows a high performance, high integration configuration of the 82588 with the 80188 in a typical iAPX188-based microcomputer. The 80188 controls the 82588, as well as providing DMA control services for data transfer, using its "on chip" two channel DMA controller.

10.1 Link Interface

The Serial Interface Mode configuration parameter selects either a highly integrated Direct Link Interface (High Integration Mode) or a highly flexible Transceiver Interface (High Speed Mode).

10.2 Application

In the High Integration mode it is possible to connect the 82588 on a very short "Wired OR" link, on a longer twisted pair cable, or a broadband connection.

TWISTED PAIR CONNECTION

The link consists of a twisted pair that interconnects the 82588. The transmit data pin is connected via a driver and the receive data pin is connected via a buffer. The twisted pair must be properly terminated to prevent reflections.

In the minimum configuration, TXD and RXD are connected to the twisted pair and CTS is grounded. The 82588 may control the driver with the RTS pin. It is also possible to use external circuitry for performing collision detection and feeding it to the 82588 through the CDT pin.

BROADBAND CONNECTION

The 82588 supports data communications over a broadband link in both its modes. Proper MODEM interface should be provided. Collision Detection by Bit Comparison, in High Integration Mode, fits transmission over broadband links.

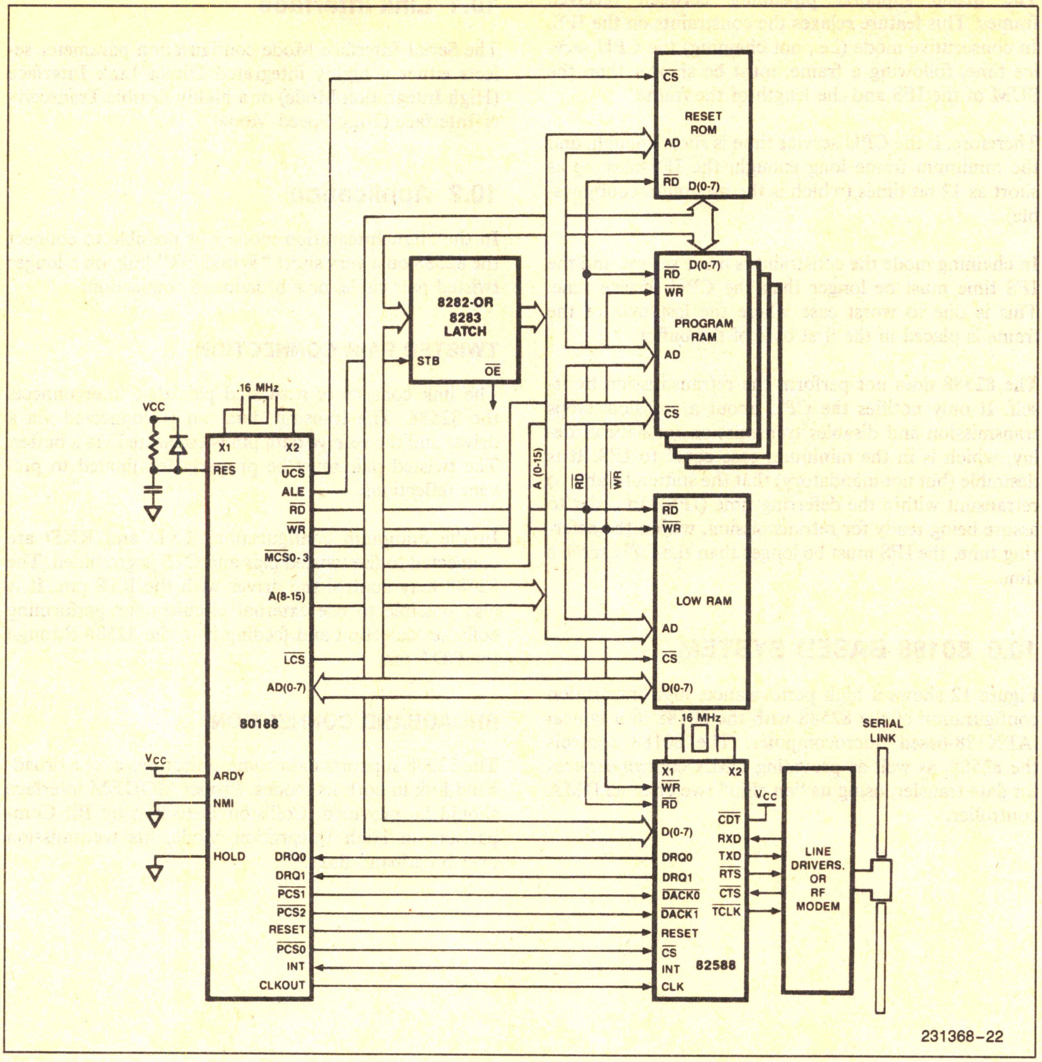


Figure 12. 80188 Based System

APPENDIX A: SOFTWARE

This section covers the basic software procedures to drive the 82588. It is written for an 80186-82588 system where both the DMA channels of the 80186 are used. The on-chip interrupt controller of the 80186 is also used.

This is not a complete software driver for the 82588, but it gives all the procedures needed to write a complete driver.

Figure A-1 shows the way of assigning the port definitions for an 80186 based system.

Figure A-2 shows how to start the various operations possible with the chip. This is not a procedure, but just examples of calls to operations.

Figure A-3 shows how the configuration parameters, individual address and multicast address parameters can be filled in the buffers before issuing the commands to program these.

Figure A-4 shows the different commands for transmit, receive, configure, etc. are issued. They also show what needs to be set up before issuing the commands.

Figure A-5 shows how the DMA controller is loaded and initialized for data and parameter transfer. This procedure is used by all other procedures requiring DMA service.

Figure A-6 shows an interrupt service routine which handles interrupts resulting due to various possible events. It also shows how to extract receive status from a received frame and retransmit sequence required after a collision.

```
/* port definitions */

  declare pba        literally '400h';        /* 80186 i/o base address */
  declare pcs0       literally 'pba + (0 * 80h)';
  declare pcs1       literally 'pba + (1 * 80h)';
  declare pcs2       literally 'pba + (2 * 80h)';
  declare pcs3       literally 'pba + (3 * 80h)';
  declare pcs4       literally 'pba + (4 * 80h)';
  declare pcs5       literally 'pba + (5 * 80h)';
  declare pcs6       literally 'pba + (6 * 80h)';

  declare cs_588     literally 'pcs1 + 0',    /* 82588 command/status */
          ch_a_588   literally 'pcs2 + 0',    /* 82588 DMA channel A */
          ch_b_588   literally 'pcs3 + 0';    /* 82588 DMA channel B */
```

231368-24

Figure A-1. Port Definition for 80186 Based System

```
/* typical calls to execute 82588 commands */

   call ia_set(1);                                      /* 1 */
   call config(1);                                      /* 2 */
   call multicast(1);                                   /* 3 */
   call transmit(1,100,@tx_buff_588);                   /* 4 */
   call tdr;                                            /* 5 */
   call dump_588(0);                                    /* 6 */
   call diagnose;                                       /* 7 */
   call rcv_enable(0,@buffer_588(0) rx(0));             /* 8 */
   call rcv_stop;                                       /* B */
   call retransmit;                                     /* C */
   call abort(1);                                       /* D */
   call reset_588;                                      /* E */

/*-------------------------------------------------------------------------*/
```

231368-25

Figure A-2. An Example of Executing Commands

```
/* 82588 init */

init_588:  procedure;

   config_588(00) = 10;           /* to configure all parameters */
   config_588(01) = 00;
   config_588(02) = 00101000b;    /* mode 0, 16 MHz clock, 1 MB/s */
   config_588(03) = buff_len/4;   /* Receive Buffer length = 256 bytes */
   config_588(04) = 10100110b;    /* Ext. loopback, address len = 6, Preamble = 8 */
   config_588(05) = 00000000b;    /* Differential Manchester = off  */
   config_588(06) = 48;           /* IFS = 48 TCLK */
   config_588(07) = 70h;          /* Slot time = 70h TCLK */
   config_588(08) = 11111000b;    /* Col Det By Bit Comp. = on  */
   config_588(09) = 00000100b;    /* Manchester encoded, bit-stuffing = off */
   config_588(10) = 10001100b;    /* Internal CRS and CDT, CRSF = 4 */
   config_588(11) = 06;

   ia_set_buff_588(0) = 6;        /* 6 byte individual address */
   ia_set_buff_588(1) = 0;
   ia_set_buff_588(2) = 000h;
   ia_set_buff_588(3) = 011h;
   ia_set_buff_588(4) = 022h;
   ia_set_buff_588(5) = 044h;
   ia_set_buff_588(6) = 088h;
   ia_set_buff_588(7) = 0ffh;

   multicast_buff_588(00) = 12;   /* Two 6 byte multicast addresses */
   multicast_buff_588(01) = 00h;
   multicast_buff_588(02) = 11h;
   multicast_buff_588(03) = 12h;
   multicast_buff_588(04) = 13h;
   multicast_buff_588(05) = 14h;
   multicast_buff_588(06) = 15h;
   multicast_buff_588(07) = 16h;
   multicast_buff_588(08) = 21h;
   multicast_buff_588(09) = 22h;
   multicast_buff_588(10) = 23h;
   multicast_buff_588(11) = 24h;
   multicast_buff_588(12) = 25h;
   multicast_buff_588(13) = 26h;

   call reset_588;

end init_588;
```

231368-26

Figure A-3. Defining Configuration, IA and Multicast Address Parameters

```
/*-----------------------------------------------------------------------*/

nop_588: procedure;              /* command - 00 */

  output (cs_588) = 00;    /* NOP */
  return;

end nop_588;

/*-----------------------------------------------------------------------*/

ia_set: procedure(channel);      /* command - 01 */

  declare channel byte;

  call dma_load(channel,1,8,@ia_set_buff_588);
  output (cs_588) = 1 or shl (channel,4);    /* ia_set */
  return;

end ia_set;

/*-----------------------------------------------------------------------*/

config: procedure(channel);      /* command - 02 */

  declare channel byte;

  call dma_load(channel,1,12,@config_588);
  output (cs_588) = 2 or shl (channel,4);    /* configure */
  return;

end config;

/*-----------------------------------------------------------------------*/

multicast: procedure(channel);   /* command - 03 */

  declare channel byte;

  call dma_load(channel,1,14,@multicast_buff_588);
  output (cs_588) = 3 or shl (channel,4);    /* multicast */
  return;

end multicast;


/*-----------------------------------------------------------------------*/

                                 /* command - 04 */

transmit: procedure(channel,buffer_len,buffer_pointer);

  declare channel byte;
  declare buffer_len word;
  declare buffer_pointer pointer;

  tx_buffer_588(00) = buffer_len mod 256;
  tx_buffer_588(01) = buffer_len / 256;

  tx_buff_ptr = buffer_pointer;   /* temporary storage */
  tx_channel = channel;
  tx_buffer_len = 2048;          /* max. frame size */

  call dma_load(tx_channel,1,tx_buffer_len,tx_buff_ptr);
  output (cs_588) = 4 or shl (tx_channel,4);    /* transmit */
  return;

end transmit;
```

231368-27

Figure A-4a. Setup and Execution of Commands

```
/*-------------------------------------------------------------------------*/

tdr: procedure;                      /* command - 05 */

  output (cs_588) = 5 ;   /* tdr */
  return;

end tdr;

/*-------------------------------------------------------------------------*/

dump_588: procedure(channel);        /* command - 06 */

  declare channel byte;

  call dma_load(channel,0,150,@dump_buff_588);
  output (cs_588) = 6 or shl (channel,4);   /* dump */
  return;

end dump_588;


/*-------------------------------------------------------------------------*/

diagnose: procedure;                 /* command - 07 */

  output (cs_588) = 7 ;   /* diagnose */
  return;

end diagnose;

/*-------------------------------------------------------------------------*/

                                     /* command - 08 */

rcv_enable: procedure(channel,buffer_ptr);

  declare channel byte;
  declare buffer_ptr pointer;

  call dma_load(channel,0,2048,buffer_ptr);
  output(cs_588)= 8 or shl (channel,4);
  return;

end rcv_enable;

/*-------------------------------------------------------------------------*/

allocate_buffer: procedure(buffer_ptr);   /* command - 09 */

  declare buffer_ptr pointer;
  declare new_channel byte;

  new_channel = not(rol(status_588(3),1)) and 00000001b;
  call dma_load(new_channel,0,buff_len,buffer_ptr);
  output(cs_588)= 9;
  return;

end allocate_buffer,
```

231368-28

Figure A-4b. Setup and Execution of Commands

```
rcv_disable: procedure;                    /* command - 10 */

   output(cs_588)= 10;
   return;

end rcv_disable;

/*-----------------------------------------------------------------------------*/

rcv_stop: procedure;                       /* command - 11 */

   output(cs_588)= 11;
   return;

end rcv_stop;

/*-----------------------------------------------------------------------------*/

retransmit: procedure;                     /* command - 12 */

   /* Parameters for this command are taken from the temporary
      storage used during the last Transmit command */

   call dma_load(tx_channel,1,tx_buffer_len,tx_buff_ptr);
   output (cs_588) = 12 or shl (tx_channel,4);    /* retransmit */
   return;

end retransmit;

/*-----------------------------------------------------------------------------*/

abort: procedure(channel);                 /* command - 13 */

   declare channel byte;

   output(cs_588)= 13 or shl (channel,4);
   return;

end abort;


/*-----------------------------------------------------------------------------*/

reset_588: procedure;                      /* command - 14 */

   output(cs_588) = 14;

   call config(1);                 /* configure on reset */

   return;

end reset_588;

/*-----------------------------------------------------------------------------*/


read_status_588: procedure;                /* command - 15 */

   output (cs_588) = 15;         /* release pointer, initial = 00 */

   status_588(0) = input (cs_588); /* refresh status register image */
   status_588(1) = input (cs_588);  /* in memory.
   status_588(2) = input (cs_588);
   status_588(3) = input (cs_588);
   return;

end read_status_588;
```

231368-29

Figure A-4c. Setup and Execution of Commands

```
dma_load: procedure(channel,direction,trans_len,buff_ptr) reentrant;

/* To load and start the 80186 DMA controller for the desired operation */

  declare dma_rx_mode   literally '1010001001000000b'; /* rx channel */
        /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

  declare dma_tx_mode   literally '0001011010000000b'; /* tx channel */
        /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */

  declare channel byte;       /* channel # */
  declare direction byte;     /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
  declare trans_len word;     /* byte count */
  declare buff_ptr pointer;   /* buffer pointer in seg:offset form */

  declare  buff_ptr_20bit dword;
  declare ptr1 pointer;
  declare (wrd based ptr1)(2) word;

  ptr1 = @buff_ptr;            /* convert buff_ptr to 20bit buff_ptr */
  buff_ptr_20bit = shl((buff_ptr_20bit := wrd(1)),4) + wrd(0);

  do case channel and 00000001b;
   do case direction and 00000001b;
     do;                       /* channel 0 , 588 to memory */
     outword(dma_0_dpl) = low (buff_ptr_20bit);
     outword(dma_0_dph) = high(buff_ptr_20bit);
     outword(dma_0_spl) = ch_a_588;
     outword(dma_0_sph) = 0;
     outword(dma_0_tc)  = trans_len;
     outword(dma_0_cw)  = dma_rx_mode or 0006h;   /* start DMA channel 0 */
     end;

     do;                       /* channel 0 , memory to 588 */
     outword(dma_0_dpl) = ch_a_588;
     outword(dma_0_dph) = 0;
     outword(dma_0_spl) = low (buff_ptr_20bit);
     outword(dma_0_sph) = high(buff_ptr_20bit);
     outword(dma_0_tc)  = trans_len;
     outword(dma_0_cw)  = dma_tx_mode or 0006h;   /* start DMA channel 0 */
     end;
   end;

   do case direction and 00000001b;
     do;                       /* channel 1 , 588 to memory */
     outword(dma_1_dpl) = low (buff_ptr_20bit);
     outword(dma_1_dph) = high(buff_ptr_20bit);
     outword(dma_1_spl) = ch_b_588;
     outword(dma_1_sph) = 0;
     outword(dma_1_tc)  = trans_len;
     outword(dma_1_cw)  = dma_rx_mode or 0006h;   /* start DMA channel 1 */
     end;

     do;                       /* channel 1 , memory to 588 */
     outword(dma_1_dpl) = ch_b_588;
     outword(dma_1_dph) = 0;
     outword(dma_1_spl) = low (buff_ptr_20bit);
     outword(dma_1_sph) = high(buff_ptr_20bit);
     outword(dma_1_tc)  = trans_len;
     outword(dma_1_cw)  = dma_tx_mode or 0006h;   /* start DMA channel 1 */
     end;
   end;
  end;
  return;

end dma_load;
```

231368-30

Figure A-5. Loading and Starting the 80186 DMA Controller

```
intr_588: procedure interrupt 13;

  declare event byte;

  call read_status_588;
  event = status_588(0) and 00001111b);

  do case event;

    event_00:   ;
    event_01:   outword(dma_1_cw)  = (dma_tx_mode or 0004h);   /* stop DMA channel 1 */
    event_02:   outword(dma_1_cw)  = (dma_tx_mode or 0004h);   /* stop DMA channel 1 */
    event_03:   outword(dma_1_cw)  = (dma_tx_mode or 0004h);   /* stop DMA channel 1 */
    event_04:   do;                  /* transmit done */
                  outword(dma_1_cw)  = (dma_tx_mode or 0004h);/* stop DMA channel 1 */
                  if (status_588(2) and 10000000b) <> 0    /* collision */
                  then if (status_588(1) and 00100000b) = 0  /* max collision */
                    then intr_588_flag = 'X';         /* retransmit */
                end;
    event_05:   ;
    event_06:   outword(dma_0_cw)  = (dma_rx_mode or 0004h);   /* stop DMA channel 0 */
    event_07:   outword(dma_0_cw)  = (dma_rx_mode or 0004h);   /* stop DMA channel 0 */
    event_08:   do;
                  call rcv_disable;

               /* extract the rx_status bytes 0,1 from the received frame */
               /* frame length is in status_588(0 & 1) */
               /* multiple buffer scheme is assumed */

                  rx_buff_off = shl(double(status_588(2)),8)
                              +  double(status_588(1)) - 2;
                  rx_buff_no = low(rx_buff_off / buff_len);
                  rx_buff_off = rx_buff_off mod buff_len;
                  /* status 0 */
                  rx_stat(0) = read_byte(@buffer_588(rx_buff_no).rx(rx_buff_off));
                  rx_buff_off = rx_buff_off + 1;
                  if rx_buff_off = buff_len  /* status across buff boundaries */
                  then do;
                    rx_buff_off = 0;
                    rx_buff_no = rx_buff_no + 1;
                      end;
                  /* status 1 */
                  rx_stat(1) = read_byte(@buffer_588(rx_buff_no).rx(rx_buff_off));

                  call rcv_enable(0,@buffer_588(0).rx(0));

                end;
    event_09:   call allocate_buffer(new_buffer);
                /* new_buffer is a procedure returning the pointer to new buffer */
    event_10:   outword(dma_0_cw)  = (dma_rx_mode or 0004h);   /* stop DMA channel 0 */
    event_11:   ;
    event_12:   do;  /* re-transmit done */
                  outword(dma_1_cw)  = (dma_tx_mode or 0004h);   /* stop DMA channel 1 */
                  if (status_588(2) and 10000000b) <> 0    /* collision */
                  then if (status_588(1) and 00100000b) = 0  /* no max collision */
                    then intr_588_flag = 'X';   /* retransmit */
                end;
    event_13:   do;  /* execution aborted */
                  outword(dma_1_cw)  = (dma_tx_mode or 0004h);   /* stop DMA channel 1 */
                  intr_588_flag = 'X';
                end;
    event_14:   ;
    event_15:   ;

  end;

  outword (eoir_186)  = 8000h;   /* non specific EOI to 80186 */

  output (cs_588) = 10000000b;   /* intack, */

  return;

end intr_588;
```

231368–31

Figure A-6. Interrupt Service Routine

intel®

APPLICATION NOTE

AP-235

November 1986

An 82586 Data Link Driver

CHARLES YAGER

Order Number: 231421-002

INTRODUCTION

This application note describes a design example of an IEEE 802.2/802.3 compatible Data Link Driver using the 82586 LAN Coprocessor. The design example is based on the "Design Model" illustrated in "Programming the 82586". It is recommended that before reading this application note, the reader clearly understands the 82586 data structures and the Design Model given in "Programming the 82586".

"Programming the 82586" discusses two basic issues in the design of the 82586 data link driver. The first is how the 82586 handler fits into the operating system. One approach is that the 82586 handler is treated as a "special kind of interface" rather than a standard I/O interface. The special interface means a special driver that has the advantage of utilizing the 82586 features to enhance performance. However the performance enhancement is at the expense of device dependent upper layer software which precludes the use of a standard I/O interface.

The second issue "Programming the 82586" discusses is which algorithms to choose for the CPU to control the 82586. The algorithms used in this data link design are taken directly from "Programming the 82586". Command processing uses a linear static list, while receive processing uses a linear dynamic list.

The application example is written in C and uses the Intel C compiler. The target hardware for the Data Link Driver is the iSBC 186/51 COMMputer, however a version of the software is also available to run on the LANHIB Demo board.

1.0 FITTING THE SOFTWARE INTO THE OSI MODEL

The application example consists of four software modules:

- Data Link Driver (DLD): drives the 82586, also known as the 82586 Handler.
- Logical Link Control (LLC): implements the IEEE 802.2 standard.
- User Application (UAP): exercises the other software modules and runs a specific application.
- C hardware support: written in assembly language, supports the Intel C compiler for I/O, interrupts, and run time initialization for target hardware.

Figure 1 illustrates how these software modules combined with the 82586, 82501 and 82502 complete the first two layers of the OSI model. The 82502 implements an IEEE 802.3 compatible transceiver, while the 82501 completes the Physical layer by performing the serial interface encode/decode function.

The Data Link Layer, as defined in the IEEE 802 standard documents, is divided into two sublayers: the Logical Link Control (LLC) and the Medium Access Control (MAC) sublayers. The Medium Access Control sublayer is further divided into the 82586 Coprocessor plus the 82586 Handler. On top of the MAC is the LLC software module which provides IEEE 802.2 compatibility. The LLC software module implements the Station Component responses, dynamic addition and deletion of Service Access Points (SAPs), and a class 1 level of service. (For more information on the LLC sublayer, refer to IEEE 802.2 Logical Link Control Draft Standard.) The class 1 level of service provides a connectionless datagram interface as opposed to the class 2 level of service which provides a connection oriented level of service similar to HDLC Asynchronous Balanced Mode.

On top of the Data Link Layer is the Upper Layer Communications Software (ULCS). This contains the Network, Transport, Session, and Presentation Layers. These layers are not included in the design example, therefore the application layer of this ap note interfaces directly to the Data Link layer.

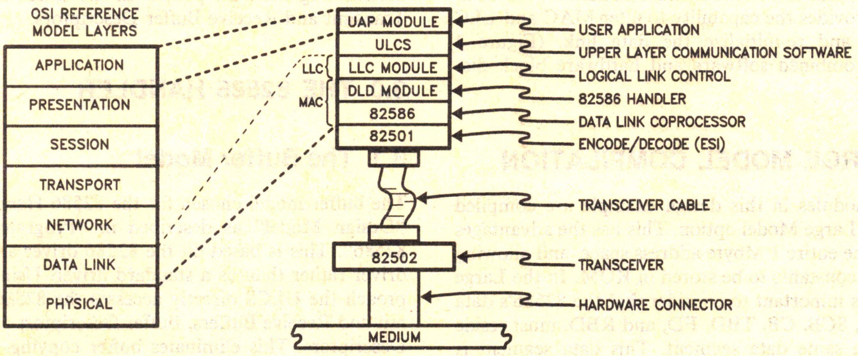


Figure 1. Data Link Driver's Relationship to OSI Reference Mode 1

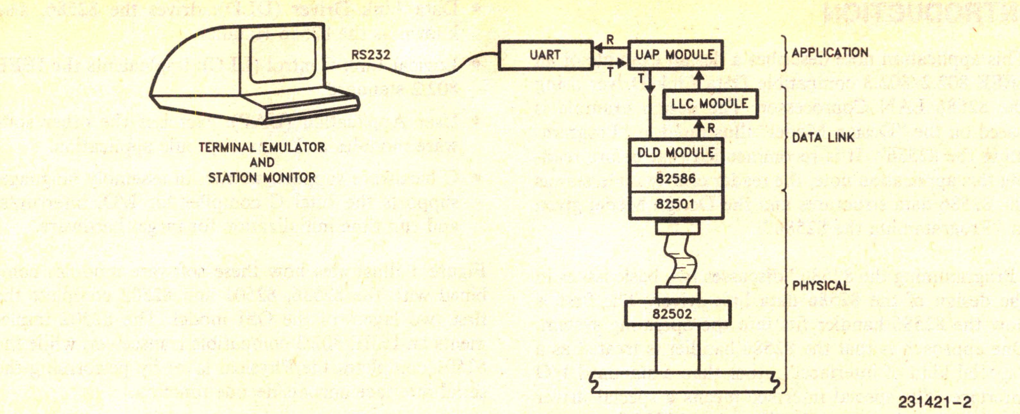


Figure 2. Block Diagram of the Hardware and Software

The application layer is implemented in the User Application (UAP) software module. The UAP module operates in one of three modes: Terminal Mode, Monitor Mode, and High Speed Transmit Mode. The software initially enters a menu driven interface which allows the program to modify several network parameters or enter one of the three modes.

The Terminal Mode implements a virtual terminal with datagram capability (connectionless "class 1" service). This mode can also be thought of as an async to IEEE 802.3/802.2 protocol converter.

The Monitor Mode provides a dynamic update on the terminal of 6 station related parameters. While in the monitor mode, any size frame can be repeatedly transmitted to the cable in a software loop.

High Speed Transmit Mode transmits frames to the cable as fast as the software possibly can. This mode demonstrates the throughput performance of the Data Link Driver.

The UAP gathers network statistics in all three modes as well as when it is in the menu. In addition, the UAP module provides the capability to alter MAC and LLC addresses and re-initialize the data link. (Figure 2 shows a combined software and hardware block diagram.)

2.0 LARGE MODEL COMPILATION

All the modules in this design example are compiled under the Large Model option. This has the advantages of using the entire 1 Mbyte address space, and allowing the string constants to be stored in ROM. In the Large Model it is important to consider that the 82586's data structures, SCB, CB, TBD, FD, and RBD, must reside within the same data segment. This data segment is determined at locate time.

The C__Assy__Support module has a run time start off function which loads the DLD data segment into a global variable SEGMT__. This data segment is used by the 82586 Handler for address translation purposes. The 82586 uses a flat address while the 80186 uses a segmented address. Any time a conversion between 82586 and 80186 addresses are needed the SEGMT__ variable is used.

Pointers for the 80186 in the large model are 32 bits, segment and offset. All the 82586 link pointers are 16 bit offsets. Therefore when trading pointers between the 82586 and the 80186, two functions are called: Offset (ptr), and Build__Ptr (offset). Offset (ptr) takes a 32 bit 80186 pointer and returns just the offset portion for the 82586 link pointer. While Build__Ptr (offset) takes an 82586 link pointer and returns a 32 bit 80186 pointer, with the segment part being the SEGMT__ variable. Offset ( ) and Build__Ptr( ) are simple functions written in assembly language included in the C__ Assy__Support module.

In the small model, Offset ( ) and Build__Ptr( ) are not needed, but the variable SEGMT__ is still needed for determining the SCB pointer in the ISCP, and in the Transmit and Receive Buffer Descriptors.

3.0 THE 82586 HANDLER

3.1 The Buffer Model

The buffer model chosen for the 82586 Handler is the "Design Model" as described in "Programming the 82586". This is based on the 82586 driver as a special driver rather than as a standard driver. Using this approach the ULCS directly accesses the 82586's Transmit and Receive Buffers, Buffer Descriptors and Frame Descriptors. This eliminates buffer copying. Transmit and receiver buffer passing is done entirely through pointers.

The only hardware dependencies between the Data Link and ULCS interface are the buffer structures. The ULCS does not handle the 82586's CBs, SCB or initialization structures. To isolate the data link interface from any hardware dependencies while still using the design model, another level of buffer copying must be introduced. For example, when the ULCS transmits a frame it would have to pass its own buffers to the data link. The data link then copies the data from ULCS buffers into 82586 buffers. When a frame is received, the data link copies the data from the 82586's buffers into the ULCS buffers. The more copying that is done the slower the throughput. However, this may be the only way to fit the data link into the operating system. The 82586 Handler can be made hardware independent by adding a receive and transmit function to perform the buffer copying.

The 82586 Handler allocates buffers from two pools of memory: the Transmit pool, and the Receive pool as illustrated in Figure 3. The Transmit pool contains Transmit Buffer Descriptors (TBDs) and Transmit Buffers (TBs). The Receive pool contains Frame Descriptors (FDs), Receive Buffer Descriptors (RBDs), and Receive Buffers (RBs).

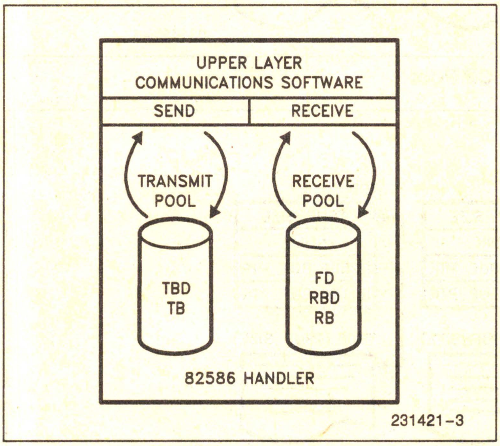


Figure 3. 82586 Handler Memory Management Model

When the ULCS wants to transmit, it requests a TBD from the handler. The handler returns a pointer to a free TBD. Each TBD has a TB attached to it. The ULCS fills the buffer, sets the appropriate fields in the TBD, and passes the TBD pointer back to the handler for transmission. After the frame is transmitted, the handler places the TBD back into the free TBD pool. If the ULCS needs more than one buffer per frame, it simply requests another TBD from the handler and performs the necessary linkage to the previous TBD.

On the receive side, the RFA pool is managed by the 82586 itself. When a frame is received, the 82586 interrupts the handler. The handler passes a FD pointer to the ULCS. Linked to the FD is one or more RBDs and RBs. The ULCS extracts what it needs from the FD, RBDs and RBs, and returns the FD pointer back to the handler. The handler places the FD and RBDs back into the free RFA pool.

3.2 The Handler Interface

The handler interface provides the following basic functions:

- initialization
- sending and receiving frames
- adding and deleting multicast addresses
- getting transmit buffers
- returning receive buffers

Figure 4 lists the Handler Interface functions.

On power up, the initialization function is called. This function initializes the 82586, and performs diagnostics. After initialization, the handler is ready to transmit and receive frames, and add and delete multicast addresses.

To send a frame, the ULCS gets one or more transmit buffers from the handler, fills them with data, and calls the send function. When a frame is received, the handler calls a receive function in the ULCS. The ULCS receive function removes the information it needs and returns the receive buffers to the handler. The addition and deletion of multicast addresses can be done "on the fly" any time after initialization. The **receiver doesn't have to be disabled** when this is done.

The command interface to the handler is totally asynchronous—the ULCS can issue transmit commands or multicast address commands whenever it wants. The commands are queued by the handler for the 82586 to execute. If the command queue is full, the send frame procedure returns a false status rather than true. The size of the command queue can be set at compile time by setting the CB—CNT constant. Typically the command queue never has more than a few commands on it because the 82586 can execute commands faster than the ULCS can issue them. This is not the case in a heavily loaded network when deferrals, collisions, and retries occur.

The command interface to the 82586 handler is hardware independent; the only hardware dependence is the buffering. A hardware independent command interface doesn't have any performance penalty, but some 82586 programmability is lost. This shouldn't be of concern since most data links do not change configuration parameters during operation. One can simply modify a few constants and recompile to change frame and network parameters to support other data links.

Handler Interface Functions	Description
Init_586( )	Initialize the Handler
Send_Frame (ptbd, padd)	Sends a frame to the cable. ptbd—Transmit Buffer Descriptor pointer padd—Destination Address pointer
Recv_Frame (pfd)	Handler calls this function which resides in the ULCS. pfd_Frame Descriptor pointer
Add_Multicast_Address (pma)	Adds one multicast address pma—Multicast Address pointer
Delete_Multicast_Address (pma)	Deletes one multicast address
Get_Tbd( )	Get a Transmit Buffer Descriptor pointer
Put_Free_Rfa (pfd)	Returns a Frame Descriptor and Receive Buffer Descriptors to the 82586.

Figure 4. List of Handler Interface Functions

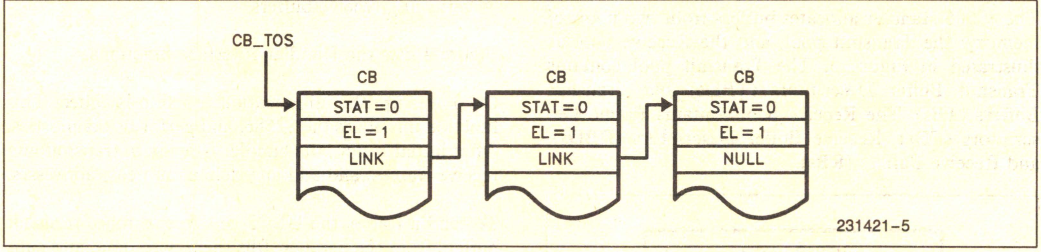


Figure 5. Free CB Pool

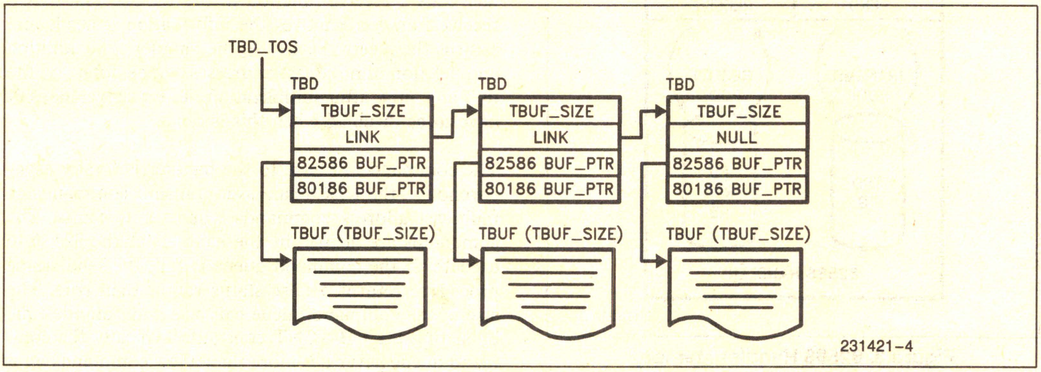


Figure 6. Free Transmit Buffer Descriptor Pool

3.3 Initialization

The function which initializes the 82586 handler, Init__586( ), is called by the ULCS on power up or reinitialization. **Before this function is called, an 82586 hardware or software reset should occur.** The Initialization occurs in three phases. The first phase is to initialize the memory. This includes flags, vectors, counters, and data structures. The second phase is to initialize the 82586. The third phase is to perform self test diagnostics. Init__586( ) returns a status byte indicating the results of the diagnostics.

Init__586( ) begins by toggling the 82501 loopback pin. If the 82501 is powered up in loopback, the $\overline{CRS}$ and $\overline{CDT}$ pin may be active. To reset this condition, the loopback pin is toggled. The 82501 should remain in loopback for the first part of the initialization function.

Phase 1 executes initialization of all the handlers flags, interrupt vectors, counters, and 82586 data structures. There are two separate functions which initialize the CB and RFA pools: Build__CB( ) and Build__Rfa( ).

3.3.1 BUILDING THE CB AND RFA POOLS

Build__CB( ) builds a stack of free linked Command Blocks, and another stack of free linked Transmit Buffer Descriptors. (See Figures 5 and 6.) Each stack has a Top of Stack pointer, which points to the next free structure. The last structure on the list has a NULL link pointer.

The CBs within the list are initialized with 0 status, EL bit set, and a link to the next CB. The TBD structures are initialized with the buffer size, which is set at compile time with the TBUF__SIZE constant, a link to the next TBD, and an 82586 pointer to the transmit buffer. This pointer is a 24 bit flat/physical address. The address is built by taking the transmit buffer's data segment address, shifting it to the left by 4 and adding it to the transmit buffer offset. An 80186 pointer to the transmit buffer is added to the TBD structure so that the 80186 does not have to translate the address each time it accesses the transmit buffer.

Build__Rfa( ) builds a linear linked Frame Descriptor list and a Receive Buffer Descriptor list as shown in Figure 7. The status and EL bits for all the free FDs are 0. The last FD's EL bit is 1 and link pointer is NULL. The first FD on the FD list points to the first RBD on the RBD list. The RBDs are initialized with both 82586 and 80186 buffer pointers. The 80186 buffer pointer is added to the end of the RBD structure. Begin and end pointers are used to mark the boundaries of the free lists.

3.3.2 82586 INITIALIZATION

The 82586 initialization data structure SCP is already set since it resides in ROM, however, the ISCP must be loaded with information. Within the SCP ROM is the pointer to the ISCP; the ISCP is the only absolute address needed in the software. Once the ISCP address is determined, the ISCP can be loaded. The SCB base is obtained from the C__Assy__Support module. The global variable SEGMT__contains the address of the

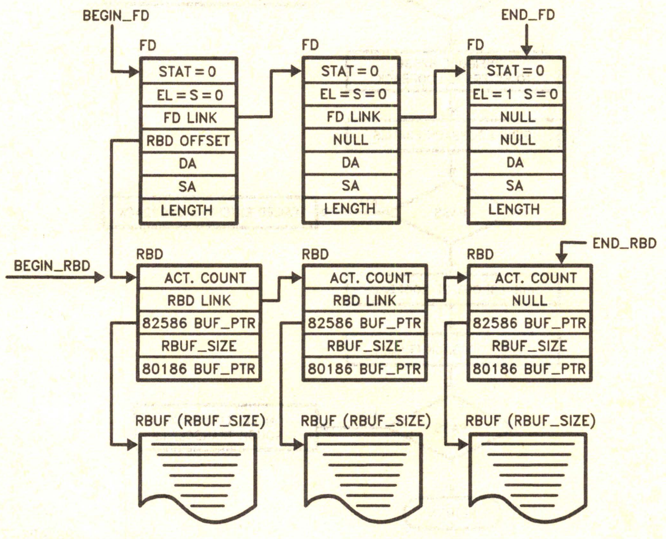


Figure 7. Free RFA

data segment of the handler. The 80186 shifts this value to the left by 4 and loads it into the SCB base. The SCB offset is now determined by taking the 32 bit SCB pointer and passing it to the Offset( ) function.

The 82586 interrupt is disabled during initialization because the interrupt function is not designed to handle 82586 reset interrupts. To determime when the 82586 is finished with its reset/initialization, the SCB status is polled for both the CX and CNA bits to be set. After the 82586 is initialized, both the CX and CNA interrupts are acknowledged.

The 82586 is now ready to execute commands. The Configuration is executed first to place the 82586 in internal loopback mode, followed by the IA command. The address for the IA command is read off of a prom on the PC board.

3.3.3 SELF TEST DIAGNOSTICS

The final phase of the handler initialization is to run the self test diagnostics. Four tests are executed: Diagnose command, Internal loopback, External loopback through the 82501, and External loopback through the transceiver. If these four tests pass, the data link is ready to go on line.

The function that executes these diagnostics is called Test_Link( ). If any of the tests fail, Test_Link( ) returns immediately with the Self_Test global variable set to the type of failure. This Self_Test global variable is then returned to the function which originally called Init_586( ). Therefore Init_586( ) can return one of five results: FAILED_DIAGNOSE, FAILED_LPBK_INTERNAL, FAILED_LPBK_EXTERNAL, FAILED_LPBK_TRANSCEIVER or PASSED.

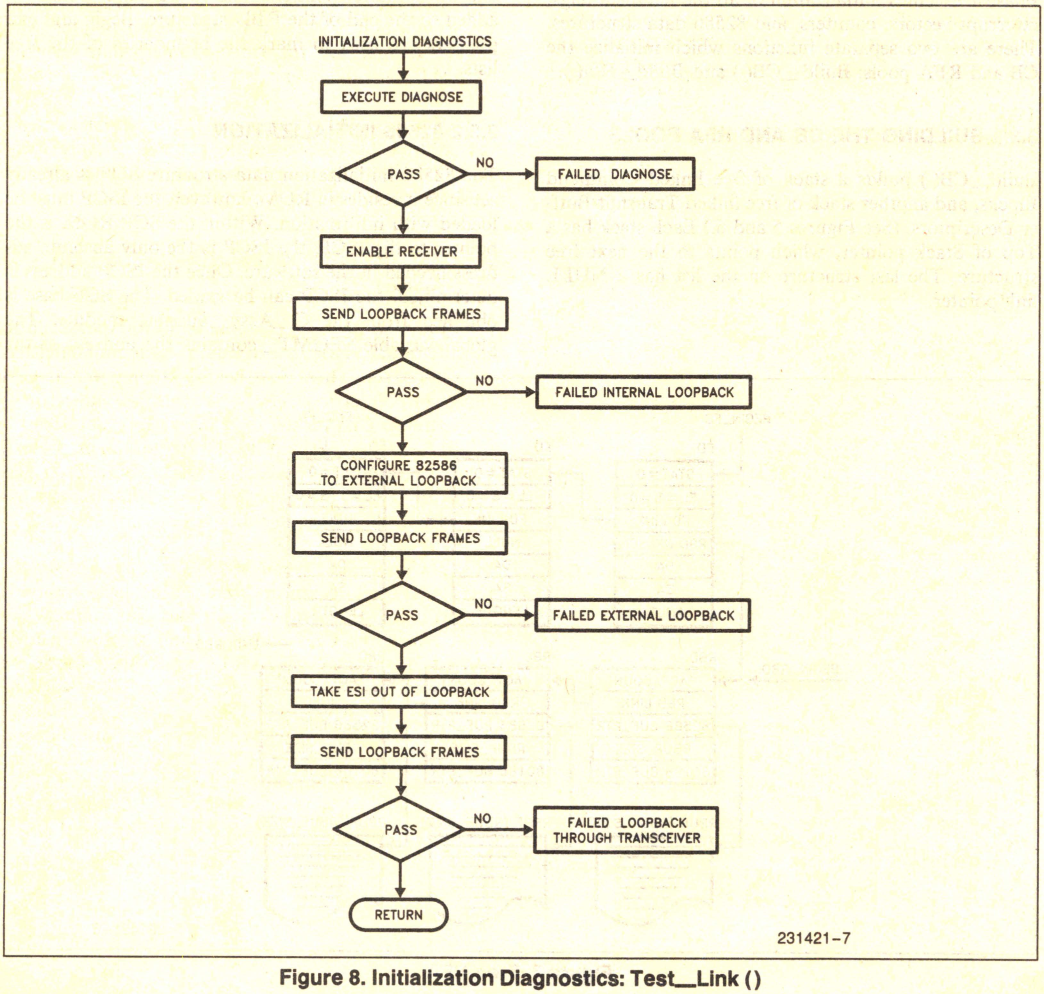


Figure 8. Initialization Diagnostics: Test_Link ()

The Diagnose( ) function, called by Test_Link( ), does not return until the diagnose command is completed. If the interrupt service routine detects that a Diagnose command was completed then it sets a flag to allow the Diagnose( ) function to return, and it also sets the Self_Test variable to FAIL if the Diagnose command failed. If the Diagnose command completed successfully, the loopback tests are performed.

Before any loopback tests are executed, the Receive Unit is enabled by calling Ru_Start( ). Loopback tests begin by calling Send_Lpbk_Frame( ), which sends 8 frames with known loopback data and its own destination address. More than one loopback frame is sent in case one or more of them are lost. Also several of the frames will have been received by the time flags.lpbk_test is checked.

Two flag bits are used for the loopback tests: flags.lpbk_mode, and flags.lpbk_test. flags.lpbk_mode is used to indicate to the receive section that the frames received are potentially loopback frames. The receive section will pass receive frames to the Loopback Check( ) function if the flags.lpbk_mode bit is set. The Loopback_Check( ) function first compares the source address of the frame with its station address. If this matches then the data is checked with the known loopback data. If the data matches, then the flags.lpbk_test bit is set, indicating a successful loopback. The flow of the Test_Link( ) function is displayed in Figure 8.

3.4 Command Processing

Command blocks are queued up on a static list for the 82586 to execute. The flow of a command block is given in Figure 9. When the handler executes a command it first has to get a free command block. It does this by calling Get_CB( ) which returns a pointer to a free command block. The CB structure is a generic one in which all commands except the MC-Setup can fit in. The handler then loads into the CB structure the type of command and associated parameters. To issue the command to the 82586 the Issue_CU_Cmd( ) function is called with the pointer to the CB passed to this function. Issue_CU_Cmd( ) places the command on the 82586's static command block list. After the 82586 executes the command, it generates an interrupt. The interrupt routine, Isr_586( ), processes the command and returns the Command Block to the free command block list by calling Put_Cb( ).

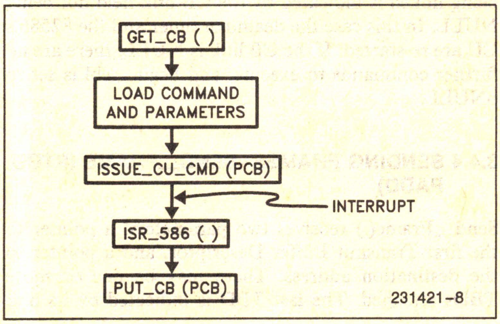


Figure 9. The Flow of a Command Block

3.4.1 ACCESSING COMMAND BLOCKS-GET_CB() and PUT_CB()

Get_Cb( ) returns a pointer to a free command block. The free command blocks are in a linear linked list structure which is treated as a stack. The pointer cb_tos points to the next available CB. Each time a CB is requested, Get_Cb( ) pops a CB off the stack. It does this by returning the pointer of cb_tos. cb_tos is then updated with the CB's link pointer. When the CB list is empty, Get_Cb( ) returns NULL.

There are two types of nulls, the 82586 'NULL' is a 16 bit offset, OFFFFH, in the 82586 data structures. The 80186 null pointer, 'pNULL', is a 32 bit pointer; with OFFFFH offset and the 82586 handler's data segment, SEGMT_, as the base.

Put_Cb( ) pushes a free command block back on the list. It does this by placing the cb_tos variable in the returned CB's link pointer field, then updates cb_tos with the pointer to the returned CB.

3.4.2 ISSUING CU COMMANDS-ISSUE_CU_CMD()

This function queues up a command for the 82586 to execute. Since static lists are used, each command has its EL bit set. There is a begin_cbl pointer and an end_cbl pointer to delineate the 82586's static list. If there are no CBs on the list, then begin_cbl is set to pNULL. (Figure 10 illustrates the static list.) Each time a command is issued, a deadman timer is set. When the 82586 interrupts the CPU with a command completed, the deadman timer is reset.

Issue_Cu_Cmd( ) begins by disabling the 82586's interrupt. It then determines whether the list is empty or not. If the list is empty, begin and end pointers are loaded with the CB's address. The CU must then be started. Before a CU_START can be issued, the SCB's cbl_offset field must be loaded with the address of the command, the Wait_Scb( ) function must be called to insure that the SCB is ready to accept a command, and the deadman timer must be initialized. If the list is not empty, then the command block is queued at the end of the list, and the interrupt service routine Isr_586( ), will continue generating CAs for each command linked on the CB list until the list is empty.

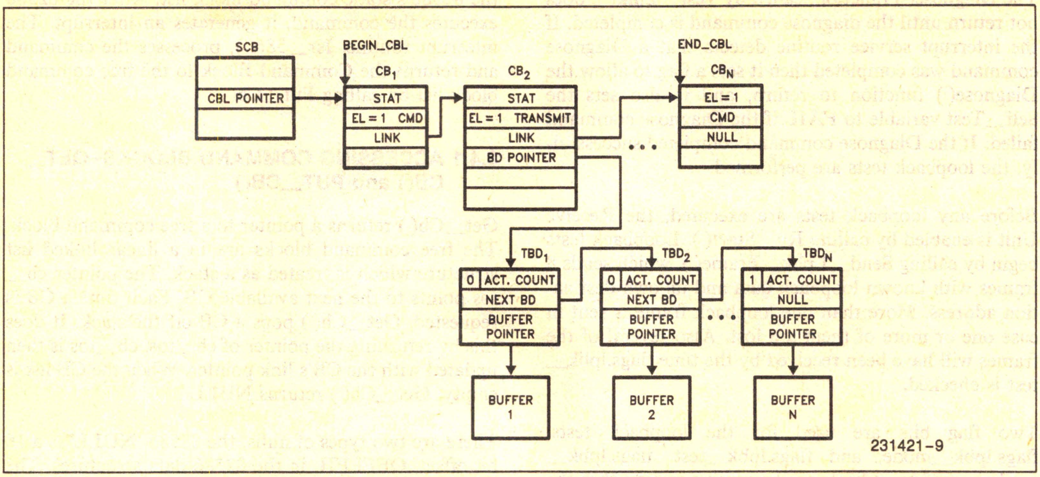


Figure 10. The Static Command Block List

3.4.3 INTERRUPT SERVICE ROUTINE-ISR__586()

Isr__586( ) starts off by saving the interrupts that were generated by the 82586 and acknowledging them. Acknowledgment must be done immediately because if a second interrupt were generated before the acknowledgment, the second interrupt would be missed. The interrupt status is then checked for a receive interrupt and if one occurred the Recv__Int__Processing( ) function is called. After receive processing is check the CPU checks whether a command interrupt occurred. If one did, then the deadman timer is reset and the results of the command are checked. There are only two particular commands which the interrupt results are checked for: Transmit and Diagnose. The Diagnose command needs to be tested to see if it passed, plus the diagnose status flag needs to be set so that the initialization process can continue.

The transmit command status provides network management and station diagnostic information which is useful for the "Network Management" function of the ISO model. The following statistics are gathered in the interrupt routine: good__transmit__cnt, sqe__err__cnt, defer__cnt, no__crs__cnt, underrun__cnt, max__col__cnt. To speed up transmit interrupt processing a flag is tested to determine whether these statistics are desired, if not this section of code is skipped.

The sqe error requires special considerations when used for statistic gathering or diagnostics. The sqe status bit indicates whether the transceiver passed its self test or not. The transceiver executes a self test after each transmission. If the transceiver's self test passed, it will activate the collision signal during the IFS time.

The sqe status bit will be set if the transceiver's self test passed. However if the sqe status bit is not set, the transceiver may still have passed its self test. Several events can prevent the sqe bit from being set. For example, the first transmit command status after power up will not have the sqe bit set because the sqe is always from the previous command. Also if any collisions occur, the sqe bit might not be set. This has to do with the timing of when the sqe signal comes from the transceiver. It is possible that a JAM signal from a remote station can overlap the sqe signal in which case the 82586 will not set the sqe status bit. Therefore the sqe error count should only be recorded when no collisions occur.

One other situation can occur which will prevent the SQE status bit from being set. If transmit command reaches the maximum retry count, the next transmit command's SQE bit will not be set.

The final phase of interrupt command processing determines if another command is linked, and returns the CB to the free command block list. Another command being linked is indicated by the CB link field not being NULL. In this case the deadman timer and the 82586's CU are re-started. If the CB link is NULL, there are no further commands to execute, and begin__cbl is set to pNULL.

3.4.4 SENDING FRAMES-SEND__FRAME (PTBD, PADD)

Send__Frame( ) receives two parameters, a pointer to the first Transmit Buffer Descriptor, and a pointer to the destination address. There may be one or more TBDs attached. The last TBD is indicated by its link

field being NULL and the EOF bit set. It is the responsibility of the ULCS to make sure this is done before calling Send__Frame( ).

Send__Frame( ) begins by trying to obtain a command block. If the free command block list is empty, the send frame function returns with a false result. It is up to the ULCS to either continue attempting transmission or attempt at a later time. The send frame function calculates the length field by summing up the TBDs actual count field. After the length field is determined, send frame checks to see if padding is required. If padding is necessary, Send Frame will change the act count field in the TBD to meet the minimum frame requirements. This technique transmits what ever was in the buffer as padding data. If security is an issue, the padding data in the buffer should be changed.

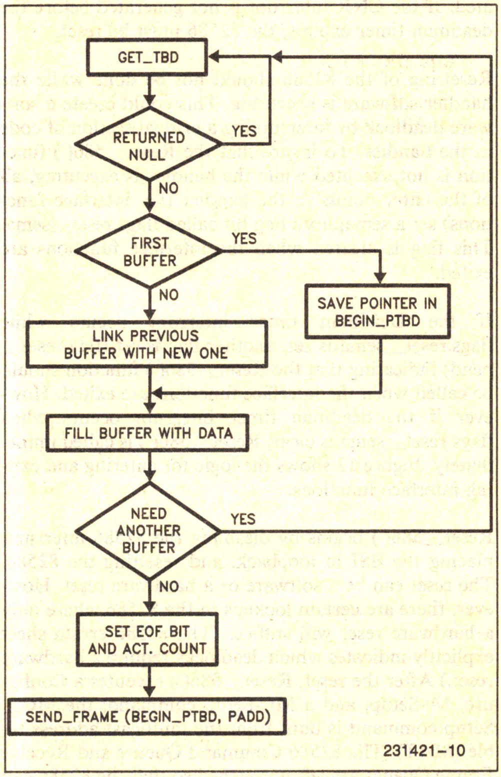


Figure 11. Flow Chart for Sending a Frame

3.4.5 ACCESSING TRANSMIT BUFFERS-GET__TBD() AND PUT__TBD()

Get__Tbd( ) returns a pointer to a free Transmit Buffer Descriptor, and Put__Tbd( ) returns one or more linked Transmit Buffer Descriptors to the free list. The TBD which Get__Tbd( ) allocates has its link pointer set to NULL, and its EOF bit cleared. If another buffer is needed, the link field in the old TBD must be set to point to the new TBD. The last TBD used should have its link pointer set to NULL and its EOF bit set. Figure 11 shows the flow chart of getting buffers and sending a frame.

Put__Tbd (ptbd) is called by the Isr__586( ) function when the 82586 is done transmitting the buffers. A pointer to the first TBD is passed to Put__Tbd( ). Put__Tbd( ) finds the end of the list of TBDs and returns them to the free buffer list.

3.4.6 MULTICAST ADDRESSES

The 82586 handler maintains a table of multicast addresses. Initially this table is empty. To enable a multicast address the Add__Multicast__Address(pma) function is called; to disable a multicast address, Delete__Multicast__Address(pma) function is called. Both functions accept a parameter which points to the multicast address. Add and Delete functions perform linear searches through the Multicast Address Table (MAT).

Add scans the entire MAT once to check if the address being added is a duplicate of one already loaded. Add will not enter a duplicate muilticast address. If there are no duplicates Add goes to the beginning of the MAT and looks for a free location. If it finds one, it loads the new address into the free location and sets the location status to INUSE. If no free locations are available, Add returns a false result.

Delete looks for a used location in the MAT. When it finds one, it compares the address in the table with the address passed to it. If they match, the location status is set to FREE and a TRUE result is returned. If no match occurs, the result returned is FALSE.

If Add or Delete change the MAT, they update the 82586 by calling Set__Multicast__Address( ). This function executes an 82586 MC Setup command. Set__Mulitcast__Address( ) uses the addresses in the MAT to build the MC Setup command. The MC Setup command is too big to be built from the free CBs. Free CB

command blocks are 18 bytes long, while the MC Setup command can be up to 16,392 bytes. Therefore a separate Multicast Address Command Block (ma__cb) must be allocated and used. The size of the ma__cb and MAT are determined at compile time based on the MULTI__ADDR__CNT constant. The design example allows up to 16 multicast addresses.

Since there is only one ma__cb, and it is not compatible with the other CBs, it must be treated differently. Only one ma__cb can be on the 82586 command list. The ma__cb command word is used as a semaphore. If it is zero, the command is available. If not, Set__Multicast__Address( ) must wait until the ma__cb is free. Also the interrupt routine can't return the ma__cb to the free CB list. It just clears the cmd field, to indicate that ma__cb is available.

The 82586's receiver does not have to be disabled to execute the MC Setup command. If the 82586 is receiving while this command is accessed, the 82586 will finish reception before executing the MC Setup comand. If the MC Setup command is executing, the 82586 automatically ignores incoming frames until the MC Setup is completed. Therefore multicast addresses can be added and deleted on the fly.

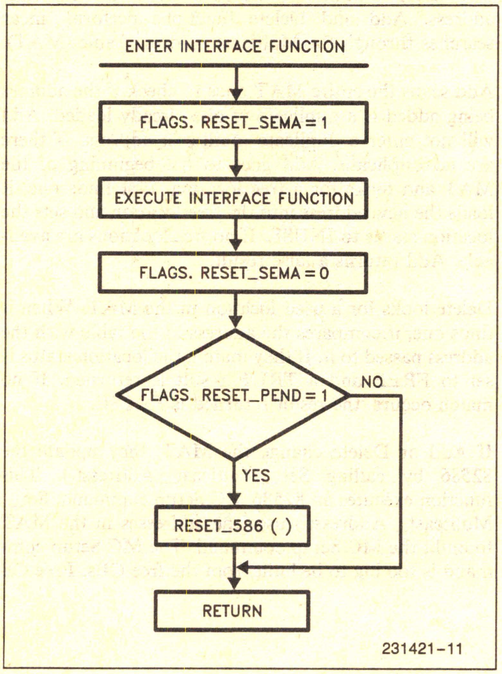


Figure 12. Reset Semaphore

3.4.7 RESETTING THE 82586-RESET__586()

The 82586 rarely if ever locks up in a well behaved network; (i.e. one that obeys IEEE 802.3 specifications). The lock-ups identified were artificially created and would normally not occur. This data link driver has been tested in an 8 station network under various loading conditions. No lock-ups occurred under any of the data link drivers test conditions. However the reset software has been tested by simulating a lockup. This can be done by having the 82586 transmit, and disabling the CTS pin for a time longer than the deadman timer.

An 82586 deadlock is not a fatal error. The handler is designed to recover from this problem. As mentioned before, each time the 82586 is given a CA to begin executing a command, a deadman timer is set. The deadman timer is reset when a CNR interrupt is generated. If the CNR interrupt is not generated before the deadman timer expires, the 82586 must be reset.

Resetting of the 82586 should not be done while the handler software is executing. This could create a software deadlock by interrupting a critical section of code in the handler. To insure that the Reset__586( ) function is not executed while the handler is executing, all of the entry points to the handler (i.e. interface functions) set a semaphore flag bit called flags.reset__sema. This flag is cleared when the interface functions are exited.

If the Deadman timer interrupt occurs while flags.reset__sema is set, another flag is set (flag.reset__pend) indicating that the Reset__586( ) function should be called when the interface functions are exited. However if the deadman timer interrupt occurs when flags.reset__sema is clear, Reset__586( ) is called immediately. Figure 12 shows the logic for entering and exiting interface functions.

Reset__586( ) begins by disabling the 82586 interrupt, placing the ESI in loopback, and resetting the 82586. The reset can be a software or a hardware reset. However, there are certain lockups in the 82586 where only a hardware reset will suffice. (The 82586 errata sheet explicitly indicates which deadlocks require a hardware reset.) After the reset, Reset__586( ) executes a Configure, IA-Setup, and a MC-Setup command; the MC__Setup command is built from the multicast address table (MAT). The 82586 Command Queues and Receive Frame Queues are left untouched so that the 82586 can continue executing where it left off before the deadlock. This way no frames or commands are lost. This requires that a separate reset CB and reset Multicast CB is used, because other CBs already in use cannot be disturbed.

3.5 Receive Frame Processing

The following functions are used for Receive Frame Processing:

Recv_Int_Processing( )	Called by Isr_586( ) to remove FDs and RBDs from the 82586's RFA
Recv_Frame (pfd)	Called by Recv_Int_Processing( ). This function resides in the ULCS
Check_Multicast (pfd)	Used for perfect Multicast filtering
Put_Free_Rfa (pfd)	Returns FDs and RBDs to the 82586's RFA
Ru_Start( )	Restarts the RU when in the IDLE or No Resources state.

3.5.1 RECEIVE INTERRUPT PROCESSING-RECV_INT_PROCESSING()

The Recv_Int_Processing( ) function is called by Isr_586( ) when the FR bit in the SCB is set. The Recv_Int_Processing( ) function checks whether any FDs and RBDs on the free list have been used by the 82586. If they have, Recv_Int_Processing( ) removes the used FDs and RBDs from the free list, and passes them to the ULCS.

The Recv_Int_Processing( ) function is a loop where each pass removes a frame from the 82586's RFA. When there are no more used FDs and RBDs on the RFA, the function calls RU_Start( ), then returns to Isr_586( ). The first part of the loop checks to see if the C bit in the first FD of the free FD list is set. If the C bit is set, the function determines if one or more RBDs are attached. If there are RBDs attached, the end of the RBD list is found. The last RBD's link field is used to update begin_rbd pointer, and then it's set to NULL.

After the receive frame has been delineated from the RFA, some information about the frame is needed to determine which function to pass it to. Since the save bad frame configure bit is not set, the only bad frame on the list could be an out of resource frame. An out of resource frame is returned to the RFA by calling Put_Free_RFA (pfd). If the flags.lpbk_mode bit is set, the frame is given to the loopback check function. If the destination address of the frame indicates a multicast, the check multicast function is called. If the frame has passed all of the above tests and still has not been returned, it is passed to the Recv_Frame( ) function which resides in the ULCS.

Check_Multicast (pfd) determines whether the multicast address received is in the multicast address table. This is necessary because the 82586 does not have perfect multicast address filtering. Check_Multicast does a byte by byte comparison of the destination address with the addresses in the multicast address table. If no match occurs, it returns false, and Recv_Int_Processing calls Put_Free_RFA( ) to return the frame to the RFA. If there is a match, Check_Multicast( ) returns TRUE and Recv_Int_Processing( ) calls Recv_Frame( ), passing the pointer to the FD of the frame received.

3.5.2 RETURNING FDs AND RBDs-PUT_FREE_RFA (pfd)

Put_Free_RFA combines Supply_FD and Supply_RBD algorithms described in "Programming the 82586" into one function. The begin and end pointers delineate what the CPU believes is the beginning and end of the free list. The decision of whether to restart the RU is made when examining both the free FD list and the free RBD list. This is why two ru_start_flags are used, one for the FD list and one for the RBD list. Both flags are initialized to FALSE.

The function starts off by initializing the FD so that the EL bit is set, the status is 0, and the FD link field is NULL. The rbd pointer is saved before the rbd pointer field in the FD is set to NULL. The free FD list is examined and if it's empty, begin—fd and end—fd are loaded with the address of the FD being returned. In this case the RU should not be restarted, because there is only one FD on the free list. If the free FD list is not empty, the FD being returned is placed on the end of the list, the end pointer is updated, and the RU start flag is set TRUE.

To begin the RBD list processing the end of the returned RBD list is determined, and this last RBD's EL bit is set. If the free RBD list is empty, the returned RBD list becomes the free RBD list. If there is more than one RBD on the returned list, the ru start flag is set TRUE. If the free RBD list is not empty, the returned RBD list is appended on the end of the free list, the end—rbd pointer is updated, and the ru start flag is set TRUE.

The last part of Put_Free_RFA( ) is to determine whether to call RU_Start( ). Both ru start flags are ANDed together, and if the result is TRUE, the Ru_Start( ) function is called.

3.5.3 RESTARTING THE RECEIVE UNIT-RU_START()

The Ru_Start( ) function checks two things before it decides to restart the RU. The first thing it checks is whether the RU is already READY. If it is, there is no reason to restart it. If the RU is IDLE or in NO_RESOURCES, then the second thing to check is whether the first free FD on the free FD list has its C bit set. If it does, then the RU should not be restarted. The reason is that the free FD list should only contain free FDs

when the RU is started. If the C bit is set in the FD, then not all the used FD have been removed yet. If the RU is started when used FDs are still in the RFA, the 82586 will write over the used FDs and frames will be lost. Therefore Ru_Start( ) is exited if the first FD in the RFA has its C bit set. If the RU is not READY, and begin_fd doesn't point to a used FD, then the RU is restarted.

Note that in "Programming the 82586" there are two more conditions to be met before the RU is started: two or more FD on the RFA, and two or more RBD on the RFA. These conditions are checked in Put_Free_RFA( ), and Ru_Start( ) isn't called unless they are met.

4.0 LOGICAL LINK CONTROL

The IEEE 802.2 LLC function completes the Data Link Layer of the OSI model. The LLC module in this design example implements a class 1 level of service which provides a connectionless datagram interface. Several data link users or processes can run on top of the data link layer. Each user is identified by a link service access point (LSAP). Communication between data link users is via LSAPs. An LSAP is an address that identifies a specific user process or another layer (see Figure 13). The LSAP addresses are defined as follows:

Data Link Layer (Station Component)	00H
Transport Layer	FEH
Network Management Layer	08H
User Processes	multiples of 4 in the range OCH < LSAP ≤ FCH

Each receiving process is identified by a destination LSAP (DSAP) and each sending process is identified by a source LSAP (SSAP). Before a destination process can receive a packet, its DSAP must be included in a list of active DSAPs for the data link.

Figure 14 illustrates the relationship between the Station Component and the SAP components. (The SAP components are user processes.) The Station Component receives all of the good frames from the Handler and checks the DSAP address. If the DSAP address is 0, then the frame is addressed to the Station Component and a Station Component Response is generated. If the DSAP address is on the active DSAP list, then the Station Component passes the frame to the addressed SAP. If the DSAP address is unknown, the frame is returned to the handler.

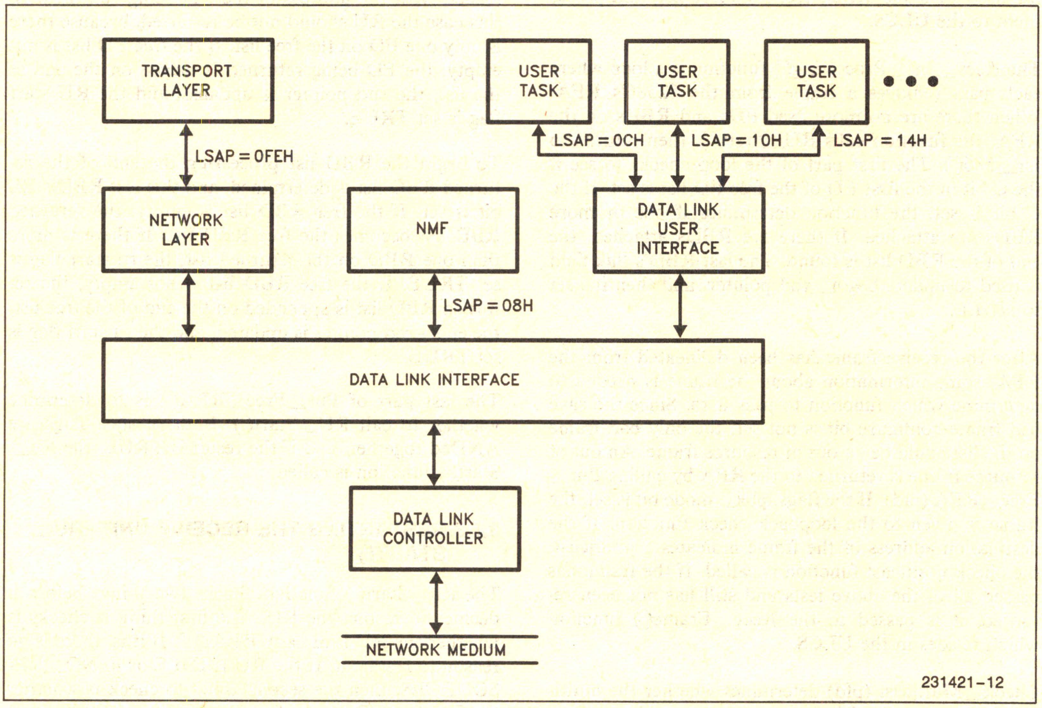


Figure 13. Data Link Interface

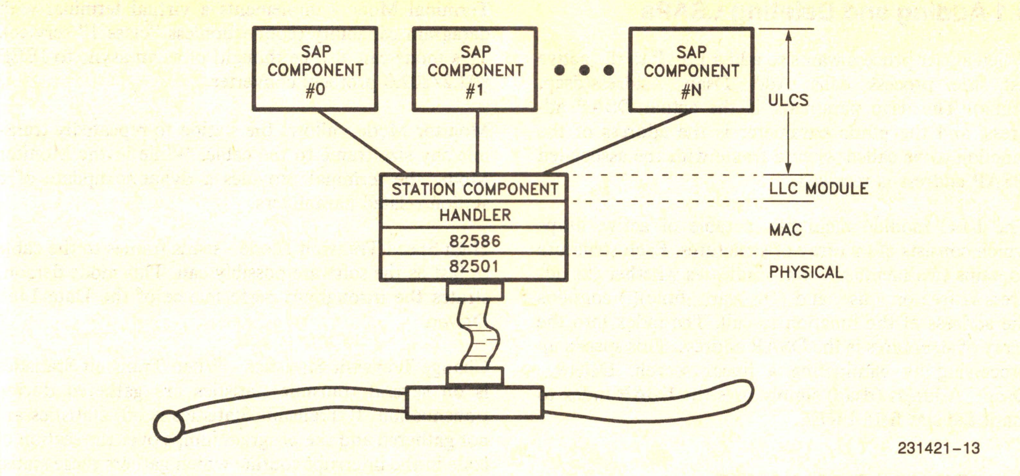


Figure 14. Station Component Relationship

There are 3 commands and 2 responses which the class 1 LLC layer must implement. Figure 15 shows IEEE 802.2 Class 1 commands and responses and Figure 16 shows the IEEE 802.2 Class 1 frame format.

Commands	Responses	Description
UI		Unnumbered Information
XID	XID	Exchange ID
TEST	TEST	Remote Loopback

Figure 15. IEEE 802.2 Class 1, Type 1 Commands and Responses

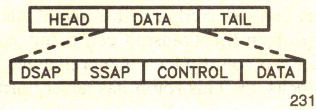


Figure 16. IEEE 802.2 Class 1 Frame Format

From Figure 15 it can be seen that there are no LLC class 1 UI responses because information frames are not acknowledged at the data link level. The only command frames that may require responses are XID and TEST. If a command frame is addressed to the Station Component, it checks the control field to see what type of frame it is. If it's an XID frame, the Station Component responds with a class 1 XID response frame. If it's a TEST frame, the Station Component responds with a TEST frame, echoing back the data it received. In both cases, the response frame is addressed to the source of the command frame.

Any frames addressed to active SAPs are passed directly to them. The Station Component will not respond to SAP addressed frames. Therefore it is the responsibility of the SAPs to recognize and respond to frames addressed to them. When a SAP transmits a frame, it builds the IEEE 802.2 frame itself and calls the Handler's Send_Frame( ) function directly. The LLC module is not used for SAP frame transmission. The only functions which the LLC module implement are the dynamic addition and deletion of DSAPs, multiplexing the frames to user SAPs, and the Station Component command recognition and responses. This is one implementation of the IEEE 802.2 standard. Other implementations may have the LLC module do more functions, such as SAP command recognitions and responses. A list of the functions included in the LLC module is as follows:

LLC Functions	Description
Init_Llc( )	Initializes the DSAP address table and calls Init_586( )
Add_Dsap_ Address (dsap, pfunc)	Add a DSAP address to the active list dsap - DSAP address pfunc - pointer to the SAP function
Delete—Dsap— Address (dsap)	Delete a DSAP address dsap - DSAP address
Recv—Frame (pfd)	Receives a frame from the 82586 Handler pfd - Frame Descriptor Pointer
Station—Component— Response (pfd)	Generates a response to a frame addressed to the Station Component pfd - Frame Descriptor Pointer

4.1 Adding and Deleting LSAPs

When a user process wants to add a LSAP to the active list, the process calls Add_Dsap_Address(dsap, pfunc). The dsap parameter is the actual DSAP address, and the pfunc parameter is the address of the function to be called when a frame with the associated DSAP address is received.

The LLC module maintains a table of active dsaps which consists of an array of structures. Each structure contains two members: stat - indicates whether the address is free or inuse, and (*p_sap_func)( ) contains the address of the function to call. The index into the array of structures is the DSAP address. This speeds up processing by eliminating a linear search. Delete_Dsap_Address (dsap) simply uses the DSAP index to mark the stat field FREE.

5.0 APPLICATION LAYER

For most networks the application layer resides on top of several other layers referred to here as ULCS. These other layers in the OSI model run from the network layer through the presentation layer. The implementation of the ULCS layers is beyond the scope of this application note, however Intel provides these layers as well as the data link layer with the OpenNET product line. For the purpose of this application note the application layer resides on top of the data link layer and its use is to demonstrate, exercise and test the data link layer design example.

There can be several processes sitting on top of the data link layer. Each process appears as a SAP to the data link. The UAP module, which implements the application layer, is the only SAP residing on top of the data link layer in this application example. Other SAPs could certainly be added such as additional "connectionless" terminals, a networking gateway, or a transport layer, however in the interest of time this was not done.

5.1 Application Layer Human Interface

The UAP provides a menu driven human interface via an async terminal connected to port B on the iSBC 186/51 board. The menu of the commands is listed in Figure 17 along with a description that follows:

Terminal Mode - implements a virtual terminal with datagram capability (connectionless "class 1" service). This mode can also be thought of as an async to IEEE 802.2/802.3 protocol converter.

Monitor Mode - allows the station to repeatedly transmit any size frame to the cable. While in the Monitor Mode, the terminal provides a dynamic update of 6 station related parameters.

High Speed Transmit Mode - sends frames to the cable as fast as the software possibly can. This mode demonstrates the throughput performance of the Data Link Driver.

Change Transmit Statistics - When Transmit Statistics is on several transmit statistics are gathered during transmission. If Transmit Statistics is off, statistics are not gathered and the program jumps over the section of code in the interrupt routine which gathers these statistics. The transmission rate is slightly increase when Transmit Statistics is off.

Print All Counters - Provides current information on the following counters.

Good frames transmitted:
Good frames received:

CRC errors received:
Alignment errors received:

Out of Resource frames:
Receiver overrun frames:

Each time a frame has been successfully transmitted the Good frames transmitted count is incremented. The same holds true for reception. CRC, Alignment, Out of Resources, and Overrun Errors are all obtained from the SCB. Underrun, lost CRS, SQE error, Max retry, and Frames that deferred are all transmit statistics that are obtained from the Transmit command status word. 82586 Reset is a count which is incremented each time the 82586 locks up. This count has never normally been incremented.

T - Terminal Mode	M - Monitor Mode
X - High Speed Transmit Mode	V - Change Transmit Statistics
P - Print All Counters	C - Clear All Counters
A - Add a Multicast Address	Z - Delete a Multicast Address
S - Change the SSAP Address	D - Change the DSAP Address
N - Change Destination Node Address	L - Print All Addresses
R - Re-Initialize the Data Link	B - Change the Number Base

Figure 17. Menu of Data Link Driver Commands

Clear All Counters - Resets all of the counters.

Add/Delete Multicast Address - Adds and Deletes Multicast Addresses.

Change SSAP Address - Deletes the previous SSAP and adds a new one to the active list. The SSAP in this case is this stations LSAP. When a frame is received, the DSAP address in the frame received is compared with any active LSAPs on the list. The SSAP is also used in the SSAP field of all transmitted frames.

Change DSAP Address - Delete the old DSAP and add a new one. The DSAP is the address of the LSAP which all transmit frames are sent to.

Change Destination Node Address - Address a new node.

Print All Addresses - Display on the terminal the station address, destination address, SSAP, DSAP, and all multicast addresses.

Re-initialize Data Link - This causes the Data Link to completely reinitialize itself. The 82586 is reset and reinitialized, and the selftest diagnostic and loopback tests are executed. The results of the diagnostics are printed on the terminal. The possible output messages from the 82586 selftest diagnostics are:

Passed Diagnostic Self Tests

Failed: Self Test Diagnose Command

Failed: Internal Loopback Self Test

Failed: External Loopback Self Test

Failed: External Loopback Through Transceiver Self Test

Change Base - Allows all numbers to be displayed in Hex or Decimal.

5.2 A Sample Session

The following text was taken directly from running the Data Link software on a 186/51 board. It begins with the iSDM monitor signing on and continues into executing the Data Link Driver software.

```
iSDM 86 Monitor, V1.0

Copyright 1983 Intel Corporation

.G D000:6

          **************************************************************
          *                                                            *
          * 82586 IEEE 802.2/802.3 Compatible Data Link Driver *
          *                                                            *
          **************************************************************

Passed Diagnostic Self Tests

Enter the Address of the Destination Node in Hex -> 00AA0000179E

Enter this Station's LSAP in Hex -> 20

Enter the Destination Node's LSAP in Hex -> 20

Do you want to Load any Multicast Addresses? (Y or N) -> Y

Enter the Multicast Address in Hex -> 00AA00111111

Would you like to add another Multicast Address? (Y or N) -> N

This Station's Host Address is: 00AA00001868

The Address of the Destination Node is: 00AA0000179E

This Station's LSAP Address is: 20

The Address of the Destination LSAP is: 20

The following Multicast Addresses are enabled: 00AA00111111
```

```
Commands are:
T - Terminal Mode                          M - Monitor Mode
X - High Speed Transmit Mode               V - Change Transmit Statistics
P - Print All Counters                     C - Clear All Counters
A - Add a Multicast Address                Z - Delete a Multicast Address
S - Change the SSAP Address                D - Change the DSAP Address
N - Change Destination Node Address        L - Print All Addresses
R - Re-Initialize the Data Link            B - Change the number Base
Enter a command, type H for Help -> P
Good frames transmitted:      24     Good frames received:        1
CRC errors received:           0     Alignment errors received:   0
Out of Resource frames:        0     Receiver overrun frames:     0
82586 Reset:                   0     Transmit underrun frames:    0
Lost CRS:                      0     SQE errors:                  9
Maximum retry:                 0     Frames that deferred:        4
Enter a command, type H for Help --> T
Would you like the local echo on? (Y or N) --> Y
This program will now enter the terminal mode.
Press ^C then CR to return back to the menu
Hello this is a test.
/*^C CR */
Enter a command, type H for Help --> M
Do you want this station to transmit? (Y or N) --> Y
Enter the number of data bytes in the frame --> 1500
Hit any key to exit Monitor Mode.
 # of Good      # of Good       CRC      Alignment     No        Receive
   Frames        Frames        Errors     Errors     Resource   Overrun
Transmitted     Received                              Errors     Errors
         32             0      00000      00000      00000      00000
/* CR */
Enter a command, type H for Help --> X
Hit any key to exit High Speed Transmit Mode.
/* CR */
Enter a command, type H for Help --> R
Passed Diagnostic Self Tests
```

5.3 Terminal Mode

The Terminal mode buffers characters received from the terminal and sends them in a frame to the cable. When a frame is received from the cable, data is extracted and sent to the terminal. One of three events initiate the UAP to send a frame providing there is data to send: buffering more than 1500 bytes, receiving a Carriage Return from the terminal, or receiving an interrupt from the virtual terminal timer.

The virtual terminal timer employs timer 1 in the 80130 to cause an interrupt every .125 seconds. Each time the interrupt occurs the software checks to see if it received one or more characters from the terminal. If it did, then it sends the characters in a frame.

The interface to the async terminal is a 256 byte software FIFO. Since the terminal communication is full duplex, there are two half duplex FIFOs: a Transmit FIFO and a Receive FIFO. Each FIFO uses two functions for I/O: Fifo_In( ) and Fifo_Out( ). A block diagram is displayed in Figure 18.

The serial I/O for the async terminal interface is always polled except in the Terminal mode where it is interrupt driven. The Terminal mode begins by enabling the 8274 receive interrupt but leaves the 8274 transmit interrupt disabled. This way any characters received from the terminal will cause an interrupt. These characters are then placed in the Transmit FIFO. The only time the 8274 transmit interrupt is enabled is when the Receive FIFO has data in it. The receive FIFO is filled from frames being received from the cable. Each time a transmit interrupt occurs a byte is removed from the Receive FIFO and written to the 8274. When the Receive FIFO empties, the 8274 transmit interrupt is disabled.

The flow control implemented for the terminal interface is via RTS and CTS. When the Transmit FIFO is full, RTS goes inactive preventing further reception of characters (see Table 1). If the Receive FIFO is full, receive frames are lost because there is no way for the data link using class 1 service to communicate to the remote station that the buffers are full. Lost receive frames are accounted for by the Out of Resources Frame counter.

The Async Terminal bit rate sets the throughput capability of the station in the terminal mode because the bottle neck for this network is the RS232 interface. Using this fact a simple test was conducted to verify the data link driver's capability of switching between the receiver's No Resource state and the Ready State. For example if station B is sending frames in the High Speed Transmit mode to station A which is in the Terminal mode, frames will be lost in station A. Under these circumstances station A's receiver will be switching from Ready state to Out of Resources state. The sum of Good frames received plus Out of Resource frames from station A should equal Good frames transmitted from station B; unless there were any underruns or overruns.

Table 1. FIFO State Table

Function	Present State	Next State	Action
FIFO_T_IN( )	EMPTY	IN USE	Start Filling Transmit Buffer
	IN USE	FULL	Shut Off RTS
FIFO_T_OUT( )	FULL	IN USE	Enable RTS
	IN USE	EMPTY	Stop Filling Transmit Buffer
FIFO_R_IN( )	EMPTY	IN USE	Turn on TxInt
	IN USE	FULL	Stop Filling FIFO from Receive Buffer
FIFO_R_OUT( )	FULL	IN USE	Start Filling FIFO from Receive Buffer
	IN USE	EMPTY	Turn Off TxInt

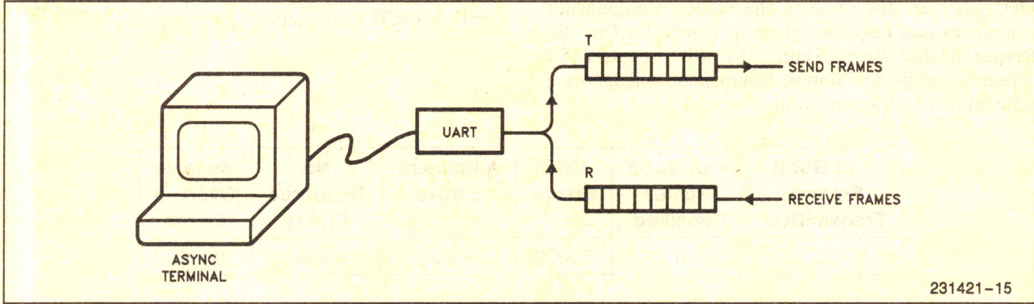


Figure 18

5.3.1 SENDING FRAMES

The Terminal Mode is entered when the Terminal__Mode( ) function is called from the Menu interface. The Terminal__Mode( ) function is one big loop, where each pass sends a frame. Receiving frames in the Terminal Mode is handled on an interrupt driven basis which will be discussed next.

The loop begins by getting a TBD from the 82586 handler. The first three bytes of the first buffer are loaded with the IEEE 802.2 header information. The loop then waits for the Transmit FIFO to become not EMPTY, at which point a byte is removed from the Transmit FIFO and placed in the TBD. After each byte is removed from the Transmit FIFO several conditions are tested to determine whether the frame needs to be transmitted, or whether a new buffer must be obtained. A frame needs to be transmitted if: a Carriage Return is received, the maximum frame length is reached, or the send__frame flag is set by the virtual terminal timer. A new buffer must be obtained if none of the above is true and the max buffer size is reached.

If a frame needs to be sent the last TBD's EOP bit is set and its buffer count is updated. The 82586 Handler's Send__Frame( ) function is called to transmit the frame, and continues to be called until the function returns TRUE.

The loop is repeated until a ^C followed by a Carriage Return is recieved.

5.3.2 RECEIVING FRAMES

Upon initialization the UAP module calls the Add__Dsap__Address(dsap, pfunc) function in the LLC module. This function adds the UAP's LSAP to the active list. The pfunc parameter is the address of the function to call when a frame has been received with the UAP's LSAP address. This function is Recv__Data__1( ). Recv__Data__1( ) looks at the control field of the frame received and determines the action required.

The commands and responses handled by Recv__Data__1( ) are the same as the Station Component's commands and responses given in Figure 15. One difference is that Recv__Data__1( ) will process a UI command while the Station Component will ignore a UI command addressed to it.

Recv__Data__1( ) will discard any UI frames received unless it is in the Terminal Mode. When in the Terminal Mode, Recv__Data__1( ) skips over the IEEE 802.2 header information and uses the length field to determine the number of bytes to place in the Receive FIFO. Before a byte is placed in the FIFO, the FIFO status is checked to make sure it is not full. Recv__Data__1( ) will move all of the data from the frame into the Receive FIFO before returning.

When a frame is received by the 82586 handler an interrupt is generated. While in the 82586 interrupt routine the receive frame is passed to the LLC layer and then to the UAP layer where the data is placed in the Receive FIFO by Recv__Octal__Data__1( ). Since Recv__Data__1( ) will not return until all of the data from the frame has been moved into the Receive FIFO, the 8274 transmit interrupt must be nested at a higher priority than the 82586 interrupt to prevent a software lock. For example if a frame is received which has more than 256 bytes of data, the Receive FIFO will fill up. The only way it can empty is if the 8274 interrupt can nest the 82586 interrupt service routine. If the 8274 could not interrupt the 82586 ISR then the software would be stuck in Recv__Data__1( ) waiting for the FIFO to empty.

5.4 Monitor Mode

The Monitor Mode dynamically updates 6 station related parameters on the terminal as shown below.

The Monitor__Mode( ) function consists of one loop. During each pass through the loop the counters are updated, and a frame is sent. Any size frame can be transmitted up to a size of the maximum number of transmit buffers available. Frame sizes less than the minimum frame length are automatically padded by the 82586 Handler.

The data in the frames transmitted in the Monitor Mode are loaded with all the printable ASCII characters. This way when one station is in the Monitor Mode transmitting to another station in the Terminal Mode, the Terminal Mode station will display a marching pattern of ASCII characters.

# of Good Frames Transmitted	# of Good Frames Received	CRC Errors	Alignment Errors	No Resource Errors	Receive Overrun Errors
32	0	00000	00000	00000	00000

5.5 High Speed Transmit Mode

The High Speed Transmit Mode demonstrates the throughput performance of the 82586 Handler. The Hs_Xmit_Mode( ) function operates in a tight loop which gets a TBD, sets the EOF bit, and calls Send_Frame( ). The flow chart for this loop is shown in Figure 19.

The loop is exited when a character is received from the terminal. Rather than polling the 8274 for a receive buffer full status, the 8274's receive interrupt is used. When the Hs_Xmit_Mode( ) function is entered, the hs_stat flag is set true. If the 8274 receive interrupt occurs, the hs_stat flag is set false. This way the loop only has to test the hs_stat flag rather than calling inb( ) function each pass through the loop to determine whether a character has been received.

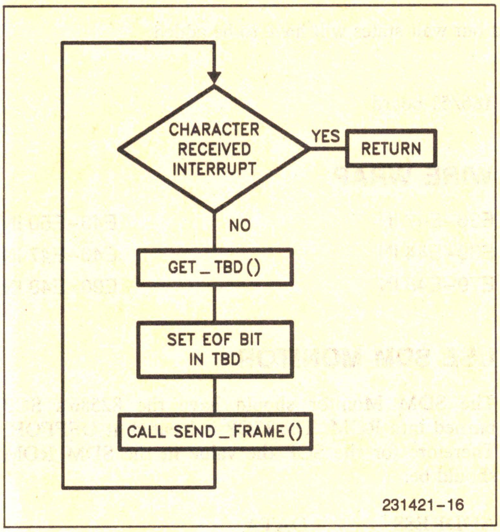


Figure 19. High Speed Transmit Mode Flow Chart

The performance measured on an 8 MHz 186/51 board is 593 frames per second. The bottle neck in the throughput is the software and not the 82586. The size of the buffer is not relevant to the transmit frame rate. Whether the buffer size is 128 bytes or 1500 bytes, linked or not, the frame rate is still the same. Therefore assuming a 1500 byte buffer at 593 frames per second, the effective data rate is 889,500 bytes per second.

This can easily be demonstrated by using two 186/51 boards running the Data Link software. The receiving stations counters should be cleared then placed in the Monitor mode. When placing it in the monitor mode, transmission should not be enabled. When the other station is placed in the High Speed Transmit Mode a timer should be started. One can use a stop watch to determine the time interval for transmission. The frame rate is determined by dividing the number of frames received in the Monitor station by the time interval of transmission.

APPENDIX A
COMPILING, LINKING, LOCATING, AND RUNNING THE SOFTWARE ON THE 186/51 BOARD

********* Instructions for using the 186/51 board *********

Use 27128A for no wait state operation. 27128s can be used but wait states will have to be added.

Copy HI.BYT and LO.BYT files into EPROMs
PROMs go into U34 - HI.BYT and U39 - LO.BYT on the 186/51 board

JUMPERS REQUIRED

Jumper the 186/51 board for 16K byte PROMs in U34 and U39 Table 2–5 in 186/51 HARDWARE REFERENCE MANUAL (Rev-001)

186/51(ES)	186/51 (S)/186/51
E151-E152 OUT	E199-E203 OUT
E152-E150 IN	E203-E191 IN
E94-E95 IN	E120-E119 IN
E100-E106 IN	E116-E112 IN
E107-E113 IN	E111-E107 IN
E133-E134 IN	E94-E93 IN

also change interrupt priority jumpers - switch 8274 and 82586 interrupt priorities

E36-E44 OUT	E43-E47 OUT
E39-E47 OUT	E46-ES0 OUT
E37-E45 OUT	E44-E48 OUT

WIRE WRAP

E36-E47 IN	E43-E50 IN
E39-E44 IN	E46-E47 IN
E79-E45 IN	E90-E48 IN

USE SDM MONITOR

The SDM Monitor should have the 82586's SCP burned into ROM. The ISCP is located at OFFFOH. Therefore for the SCP the value in the SDM ROM should be:

ADDRESS	DATA
FFFF6H	XXOOH
FFFF8H	XXXXH
FFFFAH	XXXXH
FFFFCH	FFFOH
FFFFEH	XXOOH

To run the program begin execution at 0D000:6H

```
I.E. G D000:6

GOOD LUCK!

**********    submit file for compiling one module:    **********

run

cc86.86   :F6:%0 LARGE ROM DEBUG DEFINE(DEBUG) include(:F6:)

exit

**********    submit file for linking and locating:    **********

run

link86      :F6:assy.obj, :F6:dld.obj, :F6:llc.obj, &

:F6:uap.obj, lclib.lib to :F6:dld.lnk segsize(stack(4000h)) notype

loc86 :F6:dld.lnk to :F6:dld.loc&

initcode (0D0000H) start(begin) order(classes(data, stack, code)) &
addresses(classes(data(3000H), stack(0CB00H), code(0D0020H)))

oh86 :F6:dld.loc to :F6:dld.rom

exit

**********    submit file for burning EPROMs using IPPS:    **********

ipps

i 86

f :F6:dld.rom (0d0000h)

3

2

1

0 to :F6:lo.byt

y

1 to :F6:hi.byt

y


t 27128

9

c :F6:lo.byt t p

n

C :F6:hi.byt t p

n

exit
```

```
/PCD/USR/CHUCK/CSRC/DLD.H

/***************************************************************************
*                                                                          *
*                      82586 Structures and Constants                      *
*                                                                          *
***************************************************************************/

/* general purpose constants */

#define INUSE    0
#define EMPTY    1
#define FULL     2
#define FREE     1
#define TRUE     1
#define FALSE    0
#define NULL     0xFFFF

/* Define Data Structures */

#define RBUF_SIZE          128 /* receive buffer size */
#define TBUF_SIZE          128 /* transmit buffer size */
#define ADD_LEN            6
#define MULTI_ADDR_CNT     16

typedef unsigned short int u_short;

/* results from Test_Link(): loaded into Self_Test char */

#define PASSED                         0
#define FAILED_DIAGNOSE                1
#define FAILED_LPBK_INTERNAL           2
#define FAILED_LPBK_EXTERNAL           3
#define FAILED_LPBK_TRANSCEIVER 4

/* Frame Commands */
#define      UI        0x03     /* Unnumbered Information Frame */
#define      XID       0xAF     /* Exchange Identification */
#define      TEST      0xE3     /* Remote Loopback Test */
#define      P_F_BIT 0x10      /* Poll/Final Bit Position */
#define      C_R_BIT 0x01      /* Command/Response bit in SSAP */


#define      DSAP_CNT     8     /* Number of allowable DSAPs; must be a multiple
                                   of 2**N, and DSAP addresses assigned must be
                                   divisible by 2**(8-N).
                                   (i.e. the N LSBs must be 0) */

#define      DSAP_SHIFT  5     /* DSAP_SHIFTS must equal 8-N */


#define      XID_LENGTH  6     /* Number of Info bytes for XID Response frame */


/* System Configuration Pointer SCP */

   struct SCP  {
              u_short sysbus; /* 82586 bus width, 0 - 16 bits
                                 1 - 8 bits */
```

231421-17

```
/PCO/USR/CHUCK/CSRC/DLD.H

                u_short junk[2];
                u_short iscpl;   /* lower 16 bits of iscp address */
                u_short iscph;   /* upper 8 bits of iscp address */
                };

/* Intermediate System Configuration Pointer ISCP */

   struct ISCP {
                u_short busy ;   /*set to 1 by cpu before its first CA,
                         cleared by 82586 after reading */
                u_short offset  ;    /* offset of system control block */
                u_short base1 ; /* base of system control block */
                u_short base2 ;
                } ;

/* System Control Block SCB */

   struct SCB  {
                u_short stat;        /* Status word */
                u_short cmd;         /* Command word */
                u_short cbl_offset; /* Offset of first command block in CBL */
                u_short rfa_offset; /* Offset of first frame descriptor in RFA */
                u_short crc_errs;    /* CRC errors accumulated */
                u_short aln_errs;    /* Alignment errors */
                u_short rsc_errs;    /* Frames lost because of no Resources */
                u_short ovr_errs;    /* Overrun errors */
                };

/*  Command Block */

   struct  CB  {
                u_short stat;    /* Status of Command */
                u_short cmd;     /* Command */
                u_short link;    /* link field */
                u_short parm1;   /* Parameters */
                u_short parm2;
                u_short parm3;
                u_short parm4;
                u_short parm5;
                u_short parm6;
            };


/*  Multicast Address Command Block MA_CB */

   struct MA_CB{
                u_short stat;    /* Status of Command */
                u_short cmd;     /* Command */
                u_short link;    /* Link field */
                u_short mc_cnt; /* Number of MC addresses */
                char mc_addr[ADD_LEN*MULTI_ADDR_CNT]; /* MC address area */
                };


/* Transmit Buffer Descriptor TBD */

   struct TBD  {
```

231421–18

/PCO/USR/CHUCK/CSRC/DLD.H

```
                u_short act_cnt;      /* Number of bytes in buffer */
                u_short link;         /* offset to next TBD */
                u_short buff_l;       /* lower 16 bits of buffer address */
                u_short buff_h;       /* upper 8 bits of buffer address */
                struct TB *buff_ptr;      /* not used by the 586: used by the
                                 software to save address translation
                                 routine.   */
               };

/* Transmit Buffers */
    struct TB    {
                 char data [TBUF_SIZE];
                };

/* Frame Descriptor FD */

   struct  FD  {
               u_short stat;                /* Status Word of FD */
               u_short el_s;                /* EL and S bits */
               u_short link;                /* link to next FD */
               u_short rbd_offset;          /* Receive buffer descriptor offset */
               char dest_addr[ADD_LEN]; /*Destination address */
               char src_addr[ADD_LEN];  /* Source address */
               u_short length;              /* Length field */
              };

/* Receive Buffer Descriptor RBD */

   struct RBD {
               u_short act_cnt;           /* Actual number of bytes received */
               u_short link;              /* Offset to next RBD */
               u_short buff_l;            /* Lower 16 bits of buffer address */
               u_short buff_h;            /* upper 8 bits of buffer address */
               u_short size;              /* size of buffer */
               struct RB *buff_ptr;       /* not used by the 586: used by the
                                  software to save address translation
                                  routine.  */

              };


/* Receive Buffers */

   struct RB      {
                 char data[RBUF_SIZE];
                };


struct   FRAME_STRUCT
         {
              unsigned char    dsap;    /* Destination Service Access Point */
              unsigned char    ssap;    /* Source Service Access Point */
              unsigned char    cmd;     /* ISO Data Link Command */
         };

    /* LSAP Address Table */
struct   LAT {
               char     stat;             /* INUSE or FREE */
```

231421–19

```
/PCO/USR/CHUCK/CSRC/DLD.H

              int      (*p_sap_func)();/* Pointer to LSAP function; associated
                                          with dsap address */
        };


struct MAT {                      /* Multicast Address Table */
        char     stat;            /* INUSE or FREE */
        char     addr[ADD_LEN];   /* actual mc address */
        };

/* general purpose flags */

struct FLAGS {
    unsigned diag_done : 1 ;      /* diagnose command complete */
    unsigned stat_on : 1 ;        /* network diagnostic statistics on/off */
    unsigned reset_sema: 1 ;      /* don't reset when this bit is set */
    unsigned reset_pend: 1 ;      /* reset when this bit is set */
    unsigned lpbk_test: 1 ;       /* loopback test flag */
    unsigned lpbk_mode: 1 ;       /* loopback mode on/off */
       } ;


/* General purpose bits */

#define ELBIT    0x8000
#define EOFBIT   0x8000
#define SBIT     0x4000
#define IBIT     0x2000
#define CBIT     0x8000
#define BBIT     0x4000
#define OKBIT    0x2000

/* SCB patterns */

#define CX          0x8000
#define FR          0x4000
#define CNA         0x2000
#define RNR         0x1000
#define RESET       0x0080
#define CU_START    0x0100
#define RU_START    0x0010
#define RU_ABORT    0x0040
#define CU_MASK     0x0700
#define RU_MASK     0x0070
#define RU_READY    0x0040

/* 82586 Commands */

#define NOP         0x0000
#define IA          0x0001
#define CONFIGURE   0x0002
#define MC_SETUP    0x0003
#define TRANSMIT    0x0004
#define TDR         0x0005
#define DUMP        0x0006
#define DIAGNOSE    0x0007
```

231421–20

```
/PCO/USR/CHUCK/CSRC/DLD.H

/* 82586 Command and Status Masks */

#define CMD_MASK            0x0007
#define NOERRBIT            0x2000
#define COLLMASK            0x000F
#define DEFERMASK           0x0080
#define NOCRSMASK           0x0400
#define UNDERRUNMASK        0x0100
#define SQEMASK             0x0040
#define MAXCOLMASK          0x0020
#define OUT_OF_RESOURCES 0x0200

/* Configure Parameters */

#define FIFO_LIM      0x0800          /* use FIFO lim of 8 */
#define BYTE_CNT      0x000B
#define SRDY          0x0040
#define SAV_BF        0x0080
#define ADDR_LEN      0x0600          /* address length of 6 bytes */
#define AC_LOC        0x0800
#define PREAM_LEN     0x2000          /* preamble length of 8 bytes */
#define INT_LPBCK     0x4000
#define EXT_LPBCK     0x8000
#define LIN_PRIO      0x0000          /* no priority */
#define ACR           0x0000
#define BOF_MET       0x0080
#define IFS           0x6000          /* IFS time 9.6 usec */
#define SLOT_TIME     0x0200          /* slot time 51.2 usec */
#define RETRY_NUM     0xF000          /* retry number 15 */
#define PRM           0x0001
#define BC_DIS        0x0002
#define MANCHESTER    0x0004
#define TONO_CRS      0x0008
#define NCRC_INS      0x0010
#define CRC_16        0x0020
#define BT_STUFF      0x0040
#define PAD           0x0080
#define CRSF          0x0000          /* no carrier sense filter */
#define CRS_SRC       0x0800
#define CDTF          0x0000          /* no collision detect filter */
#define CDT_SRC       0x8000
#define MIN_FRM_LEN 0x0040            /* 64 bytes */
#define MIN_DATA_LEN MIN_FRM_LEN - 18    /* assumes Ethernet/IEEE 802.3
                                            frames with 6 bytes of address */
#define MAX_FRAME_SIZE  1500 - 3
```

231421-21

```
/PCO/USR/CHUCK/CSRC/DLD.C

/***********************************************************************
*                                                                     *
*                          82586 Handler                              *
*                                                                     *
***********************************************************************/


/* Define constants for storage area */

#define CB_CNT       8   /* Number of available Command Blocks */
#define FD_CNT      16   /* Number of available Frame Descriptors */
#define RBD_CNT     64   /* Number of available Receive Buffer descriptors */
#define TBD_CNT     16   /* Number of available Transmit Buffer descriptors */

/* loopback parameters passed to Configure() */

#define INTERNAL_LOOPBACK    0x4000
#define EXTERNAL_LOOPBACK    0x8000
#define NO_LOOPBACK          0x0000

#include "dld.h"             /* 586 Data Structures */


/* 186 Timer Addresses */

#define TIMER1_CTL  0xFF5E
#define TIMER1_CNT  0xFF58
#define TIMER2_CTL  0xFF66
#define TIMER2_CNT  0xFF60


/* external functions */

/* I/O */
int     inw();        /* input word : inw(address) */
void    outw();       /* output word: outw(address, value) */
void    init_intv();/* initialize the interrupt vector table */
void    enable();     /* enable 80186 interrupts */
void    disable();    /* disable 80186 interrupts */

extern  char    *Build_Ptr();

u_short       SEGMT;       /* Data segment value */
char          *pNULL;      /* NULL pointer */

/* Macro `type' of definitions */

#define CA  outw(0xC8,0)     /* the command to issue a Channel Attention */

#define ESI_LOOPBACK outw(0xCB,0) /* put the ESI in Loopback */
#define NO_ESI_LOOPBACK outw(0xCB,8) /* take the ESI out of Loopback */

#define EOI_80130          outb(0xE0,0x63)     /* End Of Interrupt */
#define TIMER1_EOI_80186 outw(0xFF22,0x04) /* EOI for Timer 1 on the 186 */
#define TIMER1_EOI_80130 outb(0xE0,0x64) /*EOI for 186's Timer1 on the 130 */
```

231421-22

```
/PCO/USR/CHUCK/CSRC/DLD.C

/******** memory  allocation ****************/

int Self_Test;        /* used for diagnostic purposes */
u_short temp;         /* temporary storage */

#define LPBK_FRAME_SIZE      4              /* loopback frame storage */
char    lpbk_frame[LPBK_FRAME_SIZE] = {
0x55, 0xAA, 0x55, 0xAA};

#define whoami_io_add    0x00F0  /* I/O address of Host Address Prom */
char    whoami[ADD_LEN];         /* Ram array where host address is stored */

/* transmission statistic variables */

unsigned long    good_xmit_cnt;
u_short          underrun_cnt;
u_short          no_crs_cnt;
unsigned long    defer_cnt;
u_short          sqe_err_cnt;
u_short          max_col_cnt;
unsigned long    recv_frame_cnt;
u_short          reset_cnt;

    /* Allocate storage for structures and buffers */

struct FLAGS flags;

/* 586 structures */

/* System Configuration Pointer: Rom Initialization */
/* struct SCP scp = {0x0000,0x0000,0x0000,0x1FF6,0x0000};   */

/* struct ISCP iscp;    Intermediate System Configuration Pointer */

struct SCB scb;     /* System Control Block */

struct CB cb[CB_CNT],    /* Command Blocks */
*cb_tos, *begin_cbl, *end_cbl;
                            /* pointer to the beginning of the free
                               command block list (cb_tos) and the
                               beginning and end of the 82586 cbl */

struct TBD tbd[TBD_CNT],   /* Transmit Buffer Descriptor */
*tbd_tos;                 /* pointer to the free Transmit buffer
                               descriptors */

struct TB tbuf[TBD_CNT];    /* Transmit Buffers */

struct FD fd[FD_CNT],       /* Frame Descriptors */
*begin_fd, *end_fd;       /* pointers to the beginning and end of
                                the free FD list */

struct RBD rbd[RBD_CNT],    /* Receive Buffer Descriptors */
```

231421-23

```
/PCO/USR/CHUCK/CSRC/DLD.C

*begin_rbd, *end_rbd;          /* pointers to the beginning and the
                                       end of the rbd list */

struct RB rbuf[RBD_CNT];     /* Receive Buffers */

struct MAT mat[MULTI_ADDR_CNT];       /* Multicast Address Table */
struct MA_CB ma_cb;                   /* Multicast Address Command Block */

/* The following structures are used only in Reset_586() function */
struct      CB      res_cb;      /* Temporary CB for reinitializing the 586 */
struct      MA_CB   res_ma_cb;   /* Temporary MA_CB for reloading Multicast */

/*  Hardware Support Functions */

Enable_586_Int()
{
     int      c;

     c = inb(0xE2);             /* read the 80130 interrupt mask register */
     outb(0xE2, 0x00F7 & c);  /* write to the 80130 interrupt mask register */
}

Disable_586_Int()
{
     int      c;

     c = inb(0xE2);
     outb(0xE2, 0x0008 | c);
}

Set_Timeout()
{
     outw(TIMER1_CNT, 0);     /* Write a 0 to Timer1 count register */
     outw(0xFF5E, 0xE009);    /* Set ENable bit in Timer1 Mode/Control register */
}

Reset_Timeout()
{
     outw(0xFF5E, 0x6009); /* Reset ENable bit in Timer1 Mode/Control register */
}

Init_Timer() /* 186's Timer 2 is a prescaler for Timer 1.  It clocks Timer 1
                every 32.7 msec.  The deadman timeout is set for 1.25 sec */
{
     outw(0xFF38, 0x000C);     /* Set Timer1 Interrupt Control register */
     outw(0xFF62, 0xFFFF);     /* set max count register for timer2 to 0FFFFH */
     outw(0xFF5A,      38);     /* set max count register A for timer 1 */
     outw(0xFF66, 0xC001);     /* Set Timer2 Mode/Control register */
     outw(0xFF5E, 0x6009);     /* Set Timer1 Mode/Control register */
     outw(0xFF28, (inw(0xFF28) & 0xFFEF)); /* Enable 186 Timer1 interrupt */
     outb(0xE2,(inb(0xE2) & 0x00EF));   /* enable 80130 interrupt from 80186 */
}

/* end hardware support functions */

Clear_Cnt()
```

231421-24

```
/PCO/USR/CHUCK/CSRC/DLD.C


{
        scb.crc_errs = 0;           /* clear 586 error statistic counters */
        scb.aln_errs = 0;
        scb.rsc_errs = 0;
        scb.ovr_errs = 0;

        good_xmit_cnt = 0;          /* init data link statistics */
        underrun_cnt = 0;
        no_crs_cnt = 0;
        defer_cnt = 0;
        sqe_err_cnt = 0;
        max_col_cnt = 0;
        recv_frame_cnt = 0;
        reset_cnt = 0;
}

Init_586()
{
        struct ISCP *piscp;
        u_short i;
        struct  MAT *pmat;

        NO_ESI_LOOPBACK;   /* Done for 82501. Inactivates CRS if powered up
                              in loopback */
        ESI_LOOPBACK;

        init_intv();       /* Initialization DLDs interrupt vectors */

        Init_Timer();

        flags.reset_sema = 0;       /* Initialize Reset Flags */
        flags.reset_pend = 0;
        flags.stat_on = 1;

        Disable_586_Int();


        piscp =  0x0000FFF0 ; /* Initialize the ISCP pointer*/
        piscp->busy = 1;
        piscp->offset =  Offset(&scb);
        piscp->base1 = SEGMT << 4;
        piscp->base2 = (SEGMT >> 12) & 0x000F ;

        pNULL = Build_Ptr(NULL);       /* build a NULL pointer - 8086 type: 32 bits */
        Build_Rfa();       /* init Receive Frame Area */
        Build_Cb(); /* init Command Block list */
        ma_cb.cmd = 0;             /* multicast address semaphore init */

        Clear_Cnt();

        scb.stat = 0;

        CA;       /* wait for the 586 to complete initialization */


        for ( i = 0; i <= 0xFF00; i++)
```

231421-25

/PCO/USR/CHUCK/CSRC/DLD.C

```
        if (scb.stat == (CX | CNA))
            break;

    if (i >0xFF00)
    Fatal("DLD:init - Did not get an interrupt after Reset/CA\n");

    /* Ack the reset Interrupt */
    scb.cmd = (CX | CNA);
    CA;
    Wait_Scb();
    Enable_586_Int();

    scb.cbl_offset = Offset(&cb[0]);     /* link scb to cb and fd lists */
    scb.rfa_offset = Offset(&fd[0]);

     /* move the prom bytes into whoami array */

    for (i = 0; i < ADD_LEN; i++)
        whoami[(ADD_LEN - 1) - i] = inb(whoami_io_add + i*2);

        /* Initialization the Multicast Address Table */

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        pmat->stat = FREE;

    Configure(INTERNAL_LOOPBACK);    /* Put 586 in internal loopback */

    SetAddress();    /* Set up the station address */

    /* run diagnostics */

    Test_Link();

    if (Self_Test != PASSED)
        return(Self_Test);

    Configure(NO_LOOPBACK); /* Configure the 82586 */

    return(Self_Test);
}

Build_Rfa()
{
    struct    FD      *pfd;
    struct    RBD     *prbd;
    struct    RB      *pbuf;
    unsigned  long    badd;

    /* Build a linear linked frame descriptor list */

    for (pfd = &fd[0]; pfd <= &fd[FD_CNT - 1]; pfd++) {

        pfd->stat = pfd->el_s = 0;
        pfd->link = Offset(pfd+1);
        pfd->rbd_offset = NULL;
    }
```

231421-26

/PGD/USR/CHUCK/CSRC/DLD.C

```
    end_fd = --pfd;                    /* point to &fd[FD_CNT - 1] */
    pfd->link = NULL;                  /* last fd link is NULL */
    pfd->el_s = ELBIT;                 /* last fd has EL bit set */
    begin_fd = pfd = &fd[0];           /* point to first fd */
    pfd->rbd_offset = Offset(&rbd[0]);   /* link first fd to first rbd */

    /* Build a linear linked receive buffer descriptor list */

    for (prbd = &rbd[0], pbuf = &rbuf[0]; prbd <= &rbd[RBD_CNT - 1];
                                                   prbd++, pbuf++) {
        badd = SEGMT << 4;
        badd += Offset(pbuf);
        prbd->buff_l = badd;
        prbd->buff_h = badd >> 16;
        prbd->buff_ptr = pbuf;

        prbd->act_cnt =  0;
        prbd->link = Offset(prbd + 1);
        prbd->size = RBUF_SIZE;
    }

    end_rbd = --prbd;
    prbd->link = NULL;             /* last rbd points to NULL */
    prbd->size |= ELBIT;           /* last rbd has el bit set */

    begin_rbd = &rbd[0];

}

Build_Cb()  /* Build a stack of free command blocks */
{
    struct  CB   *pcb;
    struct  TBD  *ptbd;
    struct  TB   *pbuf;
    unsigned     long      badd;

    for (pcb = &cb[0]; pcb <= &cb[CB_CNT - 1]; pcb++) {
        pcb->stat = 0;
        pcb->cmd = ELBIT;
        pcb->link = Offset(pcb + 1);
    }
    --pcb;
    begin_cbl = end_cbl = pNULL;
    pcb->link = NULL;
    cb_tos = &cb[0];

    /* Build a stack of transmit buffer descriptors */

    for (ptbd = &tbd[0], pbuf = &tbuf[0]; ptbd <= &tbd[TBD_CNT - 1];
                                                   ptbd++, pbuf++) {

        ptbd->act_cnt  = TBUF_SIZE;
        ptbd->link = Offset(ptbd + 1);

        badd = SEGMT << 4;
```

231421-27

/PCD/USR/CHUCK/CSRC/DLD.C

```
        badd += Offset(pbuf);
        ptbd->buff_l = badd;
        ptbd->buff_h = badd >> 16;
        ptbd->buff_ptr = pbuf;
    }

    --ptbd;
    ptbd->link = NULL;   /* last tbd link is NULL */
    tbd_tos = &tbd[0];   /* Set the Top Of the Stack */

}

    /* Get a Command Block from the free list */

struct  CB    *Get_Cb() /* return a pointer to a free command block */
{
    struct  CB *pcb;

    if (Offset(pcb = cb_tos) == NULL)
        return(pNULL);
    cb_tos = (struct CB *) Build_Ptr(pcb->link);
    pcb->link = NULL;
    return(pcb);
}

/* Put a Command Block back onto the free list */

Put_Cb(pcb)

    struct  CB *pcb;

{
    pcb->stat = 0;
    pcb->link = Offset(cb_tos);
    cb_tos = pcb;
}

struct TBD   *Get_Tbd()   /* return a pointer to a free transmit buffer
                             descriptor */
{
    struct    TBD     *ptbd;

    flags.reset_sema = 1;
    Disable_586_Int();
    if ((ptbd = tbd_tos) != pNULL) {
        tbd_tos = (struct TBD *) Build_Ptr(ptbd->link);
        ptbd->link = NULL;
    }
    Enable_586_Int();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(ptbd);
}


Put_Tbd(ptbd)
```

231421-28

/PCO/USR/CHUCK/CSRC/DLD.C

```
struct      TBD     *ptbd;
{
    struct      TBD     *p ;

    /* find the end of the tbd list returned, ptbd is the beginning */

    for (p = ptbd; p->link != NULL; p = (struct TBD *) Build_Ptr(p->link)) ;

    p->act_cnt = TBUF_SIZE;      /* clear EOFBIT and update size on last tbd */
    p->link = Offset(tbd_tos);
    tbd_tos = ptbd;
}


SetAddress()
{
    struct  CB  *pcb;

#ifdef DEBUG

    if ((pcb = Get_Cb()) == pNULL)
    Fatal("dld.c - SetAddress - couldn't get a CB\n");

#else

    pcb = Get_Cb();

#endif  /* DEBUG */

    bcopy((char *)&pcb->parm1, &whoami[0], ADD_LEN); /* move the prom
                                                        address to IA cmd */
    pcb->cmd = IA | ELBIT;

    Issue_CU_Cmd(pcb);
}


Wait_Scb()  /* wait for the scb command word to be clear */
{
    u_short     i, stat;

    for (stat = FALSE; stat == FALSE; ) {

        for (i=0; i<=0xFF00; i++)
            if (scb.cmd == 0)
                break;

        if (i > 0xFF00) {
            Bug("DLD: Scb command not clear\n");
            CA;
        }
        else
            stat = TRUE;
    }
```

231421-29

```
/PCO/USR/CHUCK/CSRC/DLD.C

}

Issue_CU_Cmd(pcb) /* Queue up a command and issue a start CU command if no
                     other commands are queued */
    struct  CB  *pcb;
{
    Disable_586_Int();
    if (begin_cbl == pNULL) {    /* if the list is inactive start CU */
        begin_cbl = end_cbl = pcb;
        scb.cbl_offset = Offset(pcb);
        Wait_Scb();
        scb.cmd = CU_START;
        Set_Timeout();                /* set deadman timer for CU */
        CA;
    }
    else {
        end_cbl->link = Offset(pcb);
        end_cbl = pcb;
    }
    Enable_586_Int();
}

Isr7()
{
    outb(0xE0, 0x67);    /* EOI 80130 */
}

Isr6()
{
    Write("\nInterrupt 6\n");
    outb(0xE0, 0x66);    /* EOI 80130 */
}

Isr5()
{
    Write("\nInterrupt 5\n");
    outb(0xE0, 0x65);    /* EOI 80130 */
}

    /* Deadman Timer Interrupt Service Routine */

Isr_Timeout()    /* Interrupt 4 */
{
    Reset_Timeout();
    if (flags.reset_sema == 1)
        flags.reset_pend = 1;
    else
        Reset_586();

    TIMER1_EOI_80186;
    TIMER1_EOI_80130;
}

    /* Interrupt 0 is Uart in UAP Module */
    /* Interrupt 2 is Timer in UAP Module */
```

231421-30

```
/PCO/USR/CHUCK/CSRC/DLD.C

Isr1()
{
    Write("\nInterrupt 1\n");
    outb(0xE0, 0x61);      /* EOI 80130 */
}

    /* 586 Interrupt service routine: Interrupt 3 */

Isr_586()
{

    u_short     stat_scb;
    struct  CB       *pcb;

    enable();   /* nesting only the uart interrupt */

    Wait_Scb();
    scb.cmd = (stat_scb = scb.stat) & (CX | CNA | FR | RNR);
    CA;

    if (stat_scb & (FR | RNR))
        Recv_Int_Processing();

    if (stat_scb & CNA) {  /* end of cb processing */

        Reset_Timeout();     /* clear deadman timer   */
        pcb = Build_Ptr(scb.cbl_offset);

#ifdef DEBUG

        if (begin_cbl == pNULL){
            Bug("DLD: begin_cbl == NULL in interrupt routine\n");
            return;
        }

        if ((pcb->stat & 0xC000) != 0x8000)
            Fatal("DLD: C bit not set or B bit set in interrupt routine\n");

#endif /* DEBUG */

        switch (pcb->cmd & CMD_MASK) {

        case TRANSMIT:

            if (flags.stat_on == 1) {/* if Transmit Statistics are collected do */

                /* if sqe bit = 0 and there were no collisions -> sqe error
                   this condition will occur on the first transmission if
                   there were no collisions, or if the previous transmit
                   command reached the max collision count, and the current
                   transmission had no collisions */

                if ((pcb->stat & (SQEMASK | MAXCOLMASK | COLLMASK)) == 0)
                    ++sqe_err_cnt;

                if (pcb->stat & DEFERMASK)
                    ++defer_cnt;
```

231421-31

/PCO/USR/CHUCK/CSRC/DLD.C

```
            if (pcb->stat & NOERRBIT)
                ++good_xmit_cnt;
            else {
                if (pcb->stat & NOCRSMASK)
                    ++no_crs_cnt;
                if (pcb->stat & UNDERRUNMASK)
                    ++underrun_cnt;
                if (pcb->stat & MAXCOLMASK)
                    ++max_col_cnt;
            }
        }
        if (pcb->parm1 != NULL)
            Put_Tbd(Build_Ptr(pcb->parm1));
        break;

    case DIAGNOSE:

        flags.diag_done = 1;
        if ((pcb->stat & NOERRBIT) == 0)
            Self_Test = FAILED_DIAGNOSE;
        break;

    default:
            ;
    }

    /* check to see if another command is queued */

    if (pcb->link == NULL)
        begin_cbl = pNULL;

    else { /* restart the CU and execute the next command on the cbl */

        begin_cbl = Build_Ptr(pcb->link);
        scb.cbl_offset = pcb->link;
        Wait_Scb();
        scb.cmd = CU_START;
        CA;
        Wait_Scb();
        Set_Timeout();          /* START deadman timer */
    }
    if ((pcb->cmd & CMD_MASK) == MC_SETUP)
        pcb->cmd = 0; /* clear MC_SETUP cmd word, this will implement a
                         lock semephore so that it won't be reused until
                         it is completed */
    else
        Put_Cb(pcb); /* Don't return MC_SETUP cmd block. It's not a
                        general purpose command block from free CB list */
  }
  disable();   /* disable cpu int so that the 586 isr will not nest */
  EOI_80130;
}
```

231421-32

```
/PCO/USR/CHUCK/CSRC/DLD.C


Recv_Int_Processing()
{
    struct      FD      *pfd;   /* points to the Frame Descriptor */
    struct      RBD     *q,     /* points to the last rbd for the frame */
                        *prbd;  /* points to the first rbd for the frame */

    for (pfd = begin_fd; pfd != pNULL; pfd = begin_fd)
        if (pfd->stat & CBIT) {
            begin_fd = (struct FD *) Build_Ptr(pfd->link);
            prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
            if (prbd != pNULL) { /* check to see if a buffer is attached */

#ifdef DEBUG
                if (prbd != begin_rbd)
                    Fatal("DLD: prbd != begin_rbd in Recv_Int_Processing\n");
#endif /* DEBUG */
                for (q = prbd; (q->act_cnt & EOFBIT) != EOFBIT;
                                        q = (struct RBD *) Build_Ptr(q->link));

                begin_rbd = (struct RBD *) Build_Ptr(q->link);
                q->link = NULL;
            }
            if (pfd->stat & OUT_OF_RESOURCES)
                Put_Free_RFA(pfd);
            else {
                /* if the DLD is in a loopback test, check the frame recv */
                if (flags.lpbk_mode == 1)
                    Loopback_Check(pfd);
                else

            /* if it's a multicast address check to see if it's
                   in the multicast address table, if not discard the frame */

              if ( ((pfd->dest_addr[0] & 01) == 01) && (!Check_Multicast(pfd)))
                    Put_Free_RFA(pfd);
              else
                {   Recv_Frame(pfd);
                    ++recv_frame_cnt;
                }
            }
        }
        else {
            Ru_Start(); /* If RU has gone into no resources, restart it */
            break;
        }
}


Loopback_Check(pfd)             /* Called by Recv_Int_Processing; checks address
                                   and data of potential loopback frame */
    struct  FD  *pfd;
{
    struct RBD  *prbd;
    struct RB   *pbuf;
```

231421-33

/PCO/USR/CHUCK/CSRC/DLD.C

```
    if ( bcmp((char *) &pfd->src_addr[0], &whoami[0], ADD_LEN) != 0 ) {
        Put_Free_RFA(pfd);
        return;
    }
    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset); /* point to receive
                                                        buffer descriptor */
    pbuf = (struct RB *) prbd->buff_ptr; /* point to receive buffer */

    if ( bcmp((char *) pbuf, &lpbk_frame[0], LPBK_FRAME_SIZE) != 0) {
        Put_Free_RFA(pfd);
        return;
    }
    flags.lpbk_test = 1;  /* passed loopback test */
    Put_Free_RFA(pfd);
}

Check_Multicast(pfd)    /* returns true if multicast address is in MAT */
    struct      FD  *pfd;
{
    struct      MAT *pmat;

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
        (bcmp((char *) &pfd->dest_addr[0], &pmat->addr[0], ADD_LEN) == 0))
        break;

    if (pmat > &mat[MULTI_ADDR_CNT - 1])
        return(FALSE);
    return(TRUE);
}

    /* Test the Link function: executes Diagnose and Loopback tests */

Test_Link()
{
    Self_Test = PASSED;
    Diagnose();
    if (Self_Test == FAILED_DIAGNOSE)
        return;
    Ru_Start();     /* start up the RU for loopback tests */
    flags.lpbk_mode = 1; /* go into loopback mode */

    flags.lpbk_test = 0; /* set looback test to false */
    Send_Lpbk_Frame();  /* internal loopback test */
    if (flags.lpbk_test == 0) {
        Self_Test = FAILED_LPBK_INTERNAL;
        flags.lpbk_mode = 0;
        return;
    }

    flags.lpbk_test = 0;
    Configure(EXTERNAL_LOOPBACK);   /* external loopback test w/ ESI in lpbk */
    Send_Lpbk_Frame();
    if (flags.lpbk_test == 0) {
        Self_Test = FAILED_LPBK_EXTERNAL;
```

231421-34

```
/PCO/USR/CHUCK/CSRC/DLD.C

            flags.lpbk_mode = 0;
            return;
        }

        flags.lpbk_test = 0;      /* external loopback test through transceiver */
        NO_ESI_LOOPBACK;
        Send_Lpbk_Frame();
        if (flags.lpbk_test == 0)
            Self_Test = FAILED_LPBK_TRANSCEIVER;

        flags.lpbk_mode = 0; /* leave loopback mode */
}

Send_Lpbk_Frame()
{
        struct    TBD     *ptbd;
        int               i;

        for (i = 0; i < 8; i++) {     /* send lpbk frame 8 times, since it's
                                         best effort delivery */

#ifdef DEBUG
            if ((ptbd = Get_Tbd()) == pNULL)
                Fatal("dld - Send_Lpbk_Frame - couldn't get a TBD\n");
#else
            ptbd = Get_Tbd();

#endif /* DEBUG */

            ptbd->act_cnt = EOFBIT | LPBK_FRAME_SIZE;
            bcopy((char *) ptbd->buff_ptr, &lpbk_frame[0], LPBK_FRAME_SIZE);

            while(!Send_Frame(ptbd, &whoami[0]));
        }
}


Diagnose()
{
        struct    CB     *pcb;

#ifdef DEBUG
            if ((pcb = Get_Cb()) == pNULL)
            Fatal("dld - Diagnose - couldn't get a CB\n");
#else
            pcb = Get_Cb();

#endif /* DEBUG */

            flags.diag_done = 0;
            Self_Test = FALSE;
            pcb->cmd = DIAGNOSE | ELBIT;

            Issue_CU_Cmd(pcb);

            while (flags.diag_done == 0) ;   /* wait for Diag cmd to finish */
```

231421-35

```
/PCO/USR/CHUCK/CSRC/DLD.C

}

Configure(loopflag)

    u_short loopflag;
{
    struct  CB   *pcb;

#ifdef DEBUG
    if ((pcb = Get_Cb()) == pNULL)
        Fatal("dld - Configure - couldn't get a CB\n");
#else
    pcb = Get_Cb();

#endif /* DEBUG */


    /* Ethernet default parameters */

    pcb->parm1 = 0x080C;
    pcb->parm2 = 0x2600 | loopflag;
    pcb->parm3 = 0x6000;
    pcb->parm4 = 0xF200;
    pcb->parm5 = 0x0000;
    if (loopflag == NO_LOOPBACK)
        pcb->parm6 = 0x0040;
    else
        pcb->parm6 = 0x0006;     /* loopback frame is less bytes than
                                    the minimum frame length */
    pcb->cmd    = CONFIGURE | ELBIT;


    Issue_CU_Cmd(pcb);
}

/* Send a frame to the cable, pass a pointer to the destination address
   and a pointer to the first transmit buffer descriptor. */

Send_Frame(ptbd, padd)  /* returns false if it can't get a Command block */
struct      TBD      *ptbd;
char                 *padd;
{
    struct      CB  *pcb;

    u_short          length;

    flags.reset_sema = 1;

    if ((pcb = Get_Cb()) == pNULL) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    pcb->parm1 = Offset(ptbd);
```

231421-36

/PCO/USR/CHUCK/CSRC/DLD.C

```
    /* move destination address to command block */
    bcopy((char *)&pcb->parm2, (char *)padd, ADD_LEN);

    /* calculate the length field by summing up all the buffers */

    for (length = 0; ptbd->link != NULL; ptbd = Build_Ptr(ptbd->link))
        length += ptbd->act_cnt;

    length += (ptbd->act_cnt & 0x3FFF);    /* add the last buffer */

    /* check to see if padding is required, do not do padding on loopback */

    /* this will not work if MIN_DATA_LEN > TBUF_SIZE */

    if ((length < MIN_DATA_LEN) &&  /* assumes a 4 byte CRC */
        (bcmp(&whoami[0], (char *)padd, ADD_LEN) != 0))

            ptbd->act_cnt = MIN_DATA_LEN | EOFBIT;

    pcb->parm5 = length;           /* length field */


    pcb->cmd = TRANSMIT | ELBIT;


    Issue_CU_Cmd(pcb);
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}


Add_Multicast_Address(pma)    /* pma - pointer to multicast address */
char    *pma;                 /* returning false means the Multicast address
                                 table is full */
{
    struct      MAT     *pmat;

    flags.reset_sema = 1;

/* if the multicast address is a duplicate of one already in the MAT,
   then return */

    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
            (bcmp( &pmat->addr[0], (char *) pma, ADD_LEN) == 0)) {
            return(TRUE);
        }


    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if (pmat->stat == FREE) {
            pmat->stat = INUSE;
            bcopy( &pmat->addr[0], (char *) pma, ADD_LEN);
            break;
```

231421-37

/PCO/USR/CHUCK/CSRC/DLD.C

```
        }

    if (pmat > &mat[MULTI_ADDR_CNT - 1]) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    Set_Multicast_Address();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}

Delete_Multicast_Address(pma)   /* returning false means the multicast address
                                   was not found */
char    *pma;
{
    struct      MAT     *pmat;

    flags.reset_sema = 1;

    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE &&
            (bcmp( &pmat->addr[0], (char *) pma, ADD_LEN) == 0)) {
            pmat->stat = FREE;
            break;
        }

    if (pmat > &mat[MULTI_ADDR_CNT - 1]) {
        flags.reset_sema = 0;
        if (flags.reset_pend == 1)
            Reset_586();
        return(FALSE);
    }

    Set_Multicast_Address();
    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
    return(TRUE);
}

Set_Multicast_Address()
{
    struct      MAT     *pmat;
    struct      MA_CB   *pma_cb;
    u_short     i;

    i = 0;
    pma_cb = &ma_cb;
    while (pma_cb->cmd != 0) ; /* if the MA_CB is inuse, wait until it's free */
    pma_cb->link = NULL;
```

231421-38

/PCO/USR/CHUCK/CSRC/DLD.C

```
    for (pmat = mat; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE) {
            bcopy( &pma_cb->mc_addr[i], &pmat->addr[0], ADD_LEN);
            i += ADD_LEN;
        }

    pma_cb->mc_cnt = i;
    pma_cb->cmd = MC_SETUP | ELBIT;

    Issue_CU_Cmd(pma_cb);

}

Put_Free_RFA(pfd)  /* Return Frame Descriptor and Receive Buffer
                      Descriptors to the Free Receive Frame Area */

    struct      FD      *pfd;
{
    struct      RBD     *prbd,  /* points to beginning of returned RBD list */
                        *q;     /* points to end of returned RBD list */
    char        ru_start_flag_fd,   /* indicates whether to restart RU */
                ru_start_flag_rbd;

    flags.reset_sema = 1;
    ru_start_flag_fd = ru_start_flag_rbd = FALSE;
    pfd->el_s = ELBIT;
    pfd->stat = 0;
    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);/* pick up the link to the rbd */
    pfd->link = pfd->rbd_offset = NULL;

    /*  Disable_586_Int(); this command is only necessary in a multitasking
    program. However in this single task environment this routine is originally
    called from isr_586(), therefore interrupts are already disabled */


    if (begin_fd == pNULL)
        begin_fd = end_fd = pfd;
    else {
        end_fd->link = Offset(pfd);
        end_fd->el_s = 0;
        end_fd = pfd;
        ru_start_flag_fd = TRUE;
    }

    if (prbd != pNULL) {            /* if there is a rbd attached to the fd then
                                    find the beginning and end of the rbd list */

        for (q = prbd; q->link != NULL; q = Build_Ptr(q->link))
            q->act_cnt = 0;

            /* now prbd points to the beginning of the rbd list and
               q points to the end of the list */

        q->size = RBUF_SIZE | ELBIT;
        q->act_cnt = 0;
```

231421-39

/PCO/USR/CHUCK/CSRC/DLD.C

```
        if (begin_rbd == pNULL) { /* if there is nothing on the list
                                     create a new list */

            begin_rbd = prbd;
            end_rbd = q;
            if (prbd != q)
                ru_start_flag_rbd = TRUE; /* if there is more than one rbd
                                             returned start the RU */
        }
        else {
                                   /* if the rbd list already exists add on
                                     the new returned rbds */
            end_rbd->link = Offset(prbd);
            end_rbd->size = RBUF_SIZE;
            end_rbd = q;
            ru_start_flag_rbd = TRUE;
        }
    }
    if (ru_start_flag_fd && ru_start_flag_rbd)
        Ru_Start();

    /*  Enable_586_Int(); if Disable_586_Int() is used above */

    flags.reset_sema = 0;
    if (flags.reset_pend == 1)
        Reset_586();
}

Ru_Start()
{
    if ((scb.stat & RU_MASK) == RU_READY) /* if the RU is already 'ready'
                                             then return */
        return;

    if ((begin_fd->stat & CBIT) == CBIT)
        return;

    begin_fd->rbd_offset = Offset(begin_rbd); /* link the beginning of the rbd
                                                 list to the first fd */
    scb.rfa_offset = Offset(begin_fd);
    Wait_Scb();
    scb.cmd = RU_START;
    CA;
}

Software_Reset()
{
    scb.cmd = RESET;
    CA;
    Wait_Scb();
}

Issue_Reset_Cmds()
{
    Wait_Scb();
    scb.cmd = CU_START;
    CA;
```

231421-40

/PCO/USR/CHUCK/CSRC/DLD.C

```
    Wait_Scb();

    outw(0xFF5E, 0) ;          /* shut off timer 1 interrupt */
    outw(TIMER1_CNT, 0);
    outw(0xFF5E, 0xC009);      /* use timer 1 without interrupt as a deadman */

    while ((inw(0xFF5E) & 0x0020) == 0)  /* if Max Cnt bit is set before CNA
                                            is set, 586 Cmd deadlocked */
        if ((scb.stat & CNA) == CNA)
            break;

    if (scb.stat & CNA != CNA)
        Fatal("DLD:Issue_Reset_Cmds - Command deadlock during reset procedure\n");

    Reset_Timeout();

    scb.cmd = CNA;    /* Acknowledge CNA interrupt */
    CA;
    Wait_Scb();
}


/* Execute a reset, Configure, SetAddress, and MC_Setup, then restart the
    Receive Unit and the Command Unit */
Reset_586()
{
    struct      MAT     *pmat;
    u_short     i;

    ++reset_cnt;
    Disable_586_Int();
    ESI_LOOPBACK;
    Software_Reset();

    scb.stat = 0;

    CA;       /* wait for the 586 to complete initialization */


    for ( i = 0; i <= 0xFF00; i++)
        if (scb.stat == (CX | CNA))
            break;


    if (i >0xFF00)
        Fatal("DLD:init - Did not get an interrupt after Software Reset\n");

    /* Ack the reset Interrupt */
    Wait_Scb();
    scb.cmd = (CX | CNA);
    CA;
    Wait_Scb();

#ifdef  DEBUG
    if ( begin_cbl == pNULL)
        Fatal("DLD: begin_cbl = NULL in Reset_586");
#endif  /* DEBUG */
```

231421-41

```
/PCO/USR/CHUCK/CSRC/DLD.C


    /* Configure the 586 */
    /* Ethernet default parameters; Configure is not necessary when using
       default parameters  */

    res_cb.link = NULL;

    res_cb.parm1 = 0x080C;
    res_cb.parm2 = 0x2600;
    res_cb.parm3 = 0x6000;
    res_cb.parm4 = 0xF200;
    res_cb.parm5 = 0x0000;
    res_cb.parm6 = 0x0040;
    res_cb.cmd   = CONFIGURE | ELBIT;

    scb.cbl_offset = Offset(&res_cb.stat);

    Issue_Reset_Cmds();

    /* Set the Individual Address */

    bcopy((char *) &res_cb.parm1, &whoami[0], ADD_LEN); /* move the prom
                                                     address to IA cmd */
    res_cb.cmd = IA | ELBIT;

    Issue_Reset_Cmds();

    /* reload the multicast addresses */

    i = res_ma_cb.stat = 0;
    res_ma_cb.link = NULL;

    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if ( pmat->stat == INUSE ) {
            bcopy( &res_ma_cb.mc_addr[i], &pmat->addr[0], ADD_LEN);
            i += ADD_LEN;
        }

    res_ma_cb.mc_cnt = i;
    res_ma_cb.cmd = MC_SETUP | ELBIT;
    scb.cbl_offset = Offset(&res_ma_cb.stat);


    Issue_Reset_Cmds();

    /* Restart the Command Unit and the Receive Unit */

    flags.reset_sema = 0;
    flags.reset_pend = 0;

    NO_ESI_LOOPBACK;

    Recv_Int_Processing();

    scb.cbl_offset = begin_cbl;
    Wait_Scb();
```

231421-42

/PCO/USR/CHUCK/CSRC/DLD.C

```
    scb.cmd = CU_START;
    Set_Timeout();        /* Set Deadman Timer */
    CA;
    Enable_586_Int();
}

/* bcopy -- byte copy routine */
bcopy(dst, src, nbytes)
char    *dst, *src;
int     nbytes;
{
    while (nbytes--) *dst++ = *src++;
}

/* bcmp -- byte compare */
bcmp(s1, s2, nbytes)
char    *s1, *s2;
int     nbytes;
{
    while (nbytes-- && *s1++ == *s2++);
    return(*--s1 - *--s2);
}
```

231421-43

```
/PCO/USR/CHUCK/CSRC/LLC.C

/******************************************************************************
*                                                                             *
*                   IEEE 802.2 Logical Link Control Layer                     *
*                           (Station Component)                               *
******************************************************************************/

#include "dld.h"

extern  char     *pNULL;

extern  struct   TBD *Get_Tbd();
extern  char         *Build_Ptr();

readonly char    xid_frame[XID_LENGTH] = { 0, 0, XID, 0x81, 0x01, 0};
/* DSAP, SSAP, XID, xid class 1 response */

struct LAT lat[DSAP_CNT];

Init_Llc()
{
    struct  LAT     *plat;

    for (plat = &lat[0]; plat <= &lat[DSAP_CNT - 1]; plat++)
        plat->stat = FREE;
    return(Init_586());
}

/* Function for adding a new DSAP */

Add_Dsap_Address(dsap, pfunc) /* DSAP must be divisible by 2**(8-N), where
                                 2**N = DSAP_CNT.  (i.e.  N LSBs must be 0).
                                 The function will return FALSE if does not
                                 meet the above requirements, or the Lsap
                                 Address Table is full, or the address has
                                 already been used.  NULL DSAP address is
                                 reserved for the Station Component */
int dsap, (*pfunc) ();
{
    struct  LAT     *plat;

    if ((dsap << (8-DSAP_SHIFT) & 0x00FF) != 0 || dsap == 0)
        return (FALSE);

    /* Check for duplicate dsaps.  */
    if ( (plat = &lat[dsap >> DSAP_SHIFT])->stat == FREE) {
            plat->stat = INUSE;
            plat->p_sap_func = pfunc;
            return (TRUE);
    }
    else
        return(FALSE);
}

/* Function for deleting DSAPs */

Delete_Dsap_Address(dsap) /* If the specified connection exists, it is severed.
                             If the connection does not exist, the command is ignored.  */
```

231421-44

```
/PCO/USR/CHUCK/CSRC/LLC.C


int dsap;
{
    lat[dsap >> DSAP_SHIFT].stat = FREE;
}

Recv_Frame(pfd)
    struct  FD                  *pfd;
{
    struct  RBD                 *prbd;
    struct  FRAME_STRUCT        *pfs;
    struct  LAT                 *plat;

    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
    pfs  = (struct FRAME_STRUCT *) prbd->buff_ptr;

    if (pfd->rbd_offset != NULL) { /* There has to be a rbd attached
                                      to the fd, or else the frame is
                                      too short. */
        if (pfs->dsap == 0) {    /* if the frame is addressed to the Station
                                 Component, then a response may be required */

            if ( !(pfs->ssap & C_R_BIT) ) {/* if the frame received is a response,
                                              instead of a command, then reject it.
                                              Because this software does not implement
                                              DUPLICATE_ADDRESS_CHECK.-> no response
                                              frames should be recv'd */
                Station_Component_Response(pfd);
            }
        }
        /* not addressed to Station Component, */
        /* check to see if the dsap addressed is active */
        else if ((pfs->dsap << (8-DSAP_SHIFT) & 0x00FF) == 0 &&
                 (plat = &lat[(pfs->dsap) >> DSAP_SHIFT])->stat == INUSE ) {
            (*plat->p_sap_func)(pfd);    /* call the function associated
                                            with the dsap received */
            return;
        }
    }
    Put_Free_RFA(pfd); /* return the pfd if not given to the user saps */
}


Station_Component_Response(pfd)

    struct  FD          *pfd;
{
    struct FRAME_STRUCT         *prfs, *ptfs;
    struct TBD                  *ptbd, *begin_ptbd, *q;
    struct RBD                  *prbd;

    prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
    prfs  = (struct FRAME_STRUCT *) prbd->buff_ptr;

    switch (prfs->cmd & ~P_F_BIT)
        {
        case    XID:
```

231421-45

/PCO/USR/CHUCK/CSRC/LLC.C

```
            while ((ptbd = Get_Tbd()) == pNULL);
            ptbd->act_cnt = EOFBIT | XID_LENGTH;
            bcopy ((char *) ptbd->buff_ptr, &xid_frame[0], XID_LENGTH);
            ptfs = (struct FRAME_STRUCT *) ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT; /* return the frame
                                                  to the sender */
            ptfs->ssap = 0;
            while(!Send_Frame(ptbd, Build_Ptr(pfd->src_addr)));
            break;

        case    TEST:

            for (prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset),
                            q = begin_ptbd = pNULL; prbd != pNULL;
                                    prbd = Build_Ptr(prbd->link)) {

                while ((ptbd = Get_Tbd()) == pNULL);
                if (q != pNULL)
                    q->link = Offset(ptbd);
                else
                    begin_ptbd = ptbd;
                ptbd->act_cnt = prbd->act_cnt;
                bcopy((char *) ptbd->buff_ptr, (char *) prbd->buff_ptr,
                                                ptbd->act_cnt & 0x3FFF);
                q = ptbd;
            }

            ptfs = (struct FRAME_STRUCT *) begin_ptbd->buff_ptr;
            ptfs->cmd = prfs->cmd;

            ptfs->dsap = prfs->ssap | C_R_BIT;   /* return the frame to
                                                    the sender */
            ptfs->ssap = 0;
            while(!Send_Frame(begin_ptbd, Build_Ptr(pfd->src_addr)));
            break;

        }
}
```

231421-46

```
/PCO/USR/CHUCK/CSRC/UAP.C

/*****************************************************************************
*                                                                           *
*                          User Application Program                         *
*                  Async to IEEE 802.2/802.3 Protocol Converter             *
*                                                                           *
*****************************************************************************/

#include "dld.h"

/* ASCII Characters */
#define ESC     0x1B
#define LF      0x0A
#define CR      0x0D
#define BS      0x08
#define BEL     0x07
#define SP      0x20
#define DEL     0x7F
#define CTL_C           0x03

/* Hardware */
#define CH_B_CTL        0x00DE
#define CH_A_CTL        0x00DC
#define CH_B_DAT        0x00DA
#define CH_A_DAT        0x00D8
#define UART_STAT_MSK     0x70

/* Interrupt cases for 8274 */
#define UART_TX_B         0
#define UART_RECV_B       0x08
#define UART_RECV_ERR_B 0x0C
#define  EXT_STAT_INT_B 0x04
#define  EXT_STAT_INT_A 0x14

char    fifo_t[256];
char    fifo_r[256];
char    wra[5], wrb[5];
unsigned        char    in_fifo_t, out_fifo_t, in_fifo_r, out_fifo_r, actual;
u_short       t_buf_stat, r_buf_stat;

char    cbuf[80];     /* Command line buffer */
char    line[81];     /* Monitor Mode display line */

unsigned      char    dsap, ssap, send_flag, local_echo;
char    Dest_Addr[ADD_LEN];
char    Multi_Addr[ADD_LEN];

int tmstat; /* terminal mode status: for leaving terminal mode */
int dhex, monitor_flag, hs_stat;          /* flags */


extern  struct TBD       *Get_Tbd();
extern  char             *Build_Ptr();

extern  struct FLAGS     flags;

extern  char    xid_frame[];
extern  char    whoami[];
```

231421-47

```
/PCO/USR/CHUCK/CSRC/UAP.C

extern  struct  MAT       mat[];
extern  struct  LAT       lat[];
extern  char    *pNULL;

extern  unsigned long     good_xmit_cnt;
extern  u_short           underrun_cnt;
extern  u_short           no_crs_cnt;
extern  unsigned long     defer_cnt;
extern  u_short           sqe_err_cnt;
extern  u_short           max_col_cnt;
extern  unsigned long     recv_frame_cnt;
extern  u_short           reset_cnt;

extern  struct  SCB       scb;

/* Macro 'type' of definitions */

#define RTS_ONB outb(CH_B_CTL,0x05);outb(CH_B_CTL,wrb[5]=wrb[5]|0x02)
#define RTS_OFFB outb(CH_B_CTL,0x05);outb(CH_B_CTL,wrb[5]=wrb[5]&0xFD)
#define RTS_ONA outb(CH_A_CTL,0x05);outb(CH_A_CTL,wra[5]=wra[5]|0x02)
#define RTS_OFFA outb(CH_A_CTL,0x05);outb(CH_A_CTL,wra[5]=wra[5]&0xFD)
#define UART_TX_DI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]&0xFD)
#define UART_TX_EI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]|0x02)
#define UART_RX_DI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]&0xE7)
#define UART_RX_EI_B outb(CH_B_CTL,0x01);outb(CH_B_CTL,wrb[1]=wrb[1]|0x10)
#define RESET_TX_INT outb(CH_B_CTL,0x28)
#define EOI_8274 outb(CH_A_CTL,0x38) /* 8274 int is IR3 on 80130 */
#define EOI_80130_8274 outb(0xE0,0x60)
#define EOI_80130_TIMER outb(0xE0,0x62)


Enable_Uart_Int()
{
      int     c;

      c = inb(0xE2);   /* read the 80130 interrupt mask register */
      outb(0xE2, 0x00FE & c); /* write to the 80130 interrupt mask register */
}

Disable_Uart_Int()
{
      int     c;

      c = inb(0xE2);
      outb(0xE2, 0x0001 | c);
}

Enable_Timer_Int()
{
      int     c;

      outb(0xEA, 125);
      outb(0xEA, 0x00); /* Timer 1 interrupts every .125 sec */
      send_flag = FALSE;
      c = inb(0xE2);   /* read the 80130 interrupt mask register */
      outb(0xE2, 0x00FB & c); /* write to the 80130 interrupt mask register */
}
```

231421-48

```
/PCO/USR/CHUCK/CSRC/UAP.C


Disable_Timer_Int()
{
	int	c;

	c = inb(0xE2);
	outb(0xE2, 0x0004 | c);
}


Co(c)
	char	c;
{
	while ( (inb(CH_B_CTL) & 4) == 0 );
	outb(CH_B_DAT, c);
}

Ci()
{
	while ( (inb(CH_B_CTL) & 1) == 0 );
	return(inb(CH_B_DAT) & 0x7F);
}

Read(pmsg, cnt, pact)
	char	*pmsg;
	unsigned char	cnt, *pact;
{
	unsigned char	i;
	char	c,	buf[200];

	for (i = c = 0; (c != CR) && (c != LF) && (i < 198); ) {
		c = Ci() & 0x7F;
		if (c == BS || c == DEL) {
			if (i > 0) {
				--i;
				Co(BS); Co(SP); Co(BS);
			}
		}
		else
			if (c >= SP) {
				Co(c);
				buf[i++] = c;
			}
		else
			if ((c == CR) || (c == LF)) {
				buf[i++] = CR;
				buf[i++] = LF;
			}
		else Co(BEL);
	}
	Co(CR); Co(LF);
	if (i > cnt)
		*pact = cnt;
	else
		*pact = i;
	for (i = 0; i < *pact ; i++)
		*pmsg++ = buf[i];
```

231421-49

```
/PCO/USR/CHUCK/CSRC/UAP.C

}

Read_Char()
{
    unsigned char   i;

    Read(&cbuf[0], 80, &actual);
    i = Skip(&cbuf[0]);
    return(cbuf[i]);
}

Write(pmsg)
    char    *pmsg;
{
    while (*pmsg != '\0') {
        if (*pmsg == '\n')
            Co(CR);
        Co(*pmsg++);
    }
}

Fatal(pmsg) /* write a message to the screen then stop */
    char    *pmsg;
{
    Write("Fatal: ");
    Write(pmsg);
    for(;;);
}

Bug(pmsg)   /* write a message to the screen then continue */
    char    *pmsg;
{
    Write("Bug: ");
    Write(pmsg);
}

Ascii_To_Char(c) /* convert ASCII-Hex to Char */
    char    c;
{
    if (('0' <= c) && (c <= '9'))
        return(c - '0');
    if (('A' <= c) && (c <= 'F'))
        return(c - 0x37);
    if (('a' <= c) && (c <= 'f'))
        return(c - 0x57);
    return(0xFF);
}

Lower_Case(c)
char    c;
{
    if (('a' <= c) && (c <= 'z'))
        return(c);
    if (('A' <= c) && (c <= 'Z'))
        return(c + 0x20);
    return(0);
}
```

231421-50

```
/PCO/USR/CHUCK/CSRC/UAP.C


Char_To_Ascii(c, ch) /* convert char to ASCII-Hex */
    unsigned char   c, ch[];
{
    unsigned char   i;

    i = (c & 0xF0) >> 4;
    if (i < 10)
        ch[0] = i + 0x30;
    else
        ch[0] = i + 0x37;
    i = (c & 0x0F) ;
    if (i < 10)
        ch[1] = i + 0x30;
    else
        ch[1] = i + 0x37;
    ch[2] = '\0';
}

Skip(pmsg)  /* skip blanks */
    char    *pmsg;
{
    int     i;

    for (i = 0; *pmsg == ' '; i++, pmsg++);
    return(i);
}

Read_Int()  /* Read a 16 bit Integer */
{
    u_short     wd, wh, wd1, wh1, j;
    char        i, done, hex, dover, hover;

    for (done = FALSE; done == FALSE; ) {
        Read(&cbuf[0], 80, &actual);
        i = Skip(&cbuf[0]);

        for (hex = dover = hover = FALSE, wd = wh = wd1 = wh1 = 0;
                        (j = Ascii_To_Char(cbuf[i])) <= 15; i++) {
            if (j > 9)
                hex = TRUE;
            wd = wd*10 + j;
            wh = wh*16 + j;
            if (wd < wd1)
                dover = TRUE;
            if (wh < wh1)
                hover = TRUE;
            wd1 = wd; wh1 = wh;
        }
        if (cbuf[i] == 'H' || cbuf[i] == 'h' || cbuf[i] == CR ||
                        cbuf[i] == LF || cbuf[i] == ' ') {
            if (cbuf[i] == 'H' || cbuf[i] == 'h')
                hex = TRUE;
            if (hex == TRUE && hover == FALSE)
                done = TRUE;
            if (hex == FALSE && dover == FALSE)
                done = TRUE;
```

231421-51

/PCO/USR/CHUCK/CSRC/UAP.C

```
            if (!done) {
                Write("\n This number is too big.\n It has to be less than 65536.\n");
                Write("\n Enter number --> ");
            }
        }
        else
            Write(" Illegal Character\n Enter a number -->");
    }
    if (hex)
        return(wh);
    return(wd);
}

Int_To_Ascii(value, base, ld, ch, width) /* convert an integer to an ASCII string */
    unsigned long   value;
    u_short         base, width;
    char        ch[], ld;
{
    u_short i, j;

    for (i = 0; i < width; i++) {
        j = value % base;
        if (j < 10)   ch[i] = j + 0x30;
        else ch[i] = j + 0x37;
        value = value / base;
    }
    for (i = width - 1; ch[i] == '0' && i > 0; i--)
        ch[i] = ld;
    ch[width] = '\0';
}

Write_Long_Int(dw, i)
    unsigned long   dw;
    u_short     i;
{
    u_short     j;
    char      ch[11];

    if (dhex)
        Int_To_Ascii(dw, 16, ' ', &ch[0], 8);
    else
        Int_To_Ascii(dw, 10, ' ', &ch[0], 10);
    for (j = 0; ch[j] != '\0'; i--, j++)
        line[i] = ch[j];
}

Write_Short_Int(w, i)
    u_short w, i;
{
    u_short j;
    char    ch[6];
    unsigned long   dw;

    dw = w;
    if (dhex)
        Int_To_Ascii(dw, 16, '0', &ch[0], 4);
    else
```

231421-52

```
/PCO/USR/CHUCK/CSRC/UAP.C

        Int_To_Ascii(dw, 10, '0', &ch[0], 5);
    for (j = 0; ch[j] != '\0'; i--, j++)
        line[i] = ch[j];
}


Yes()
{
    char    b;

    for ( ; ; ) {
        b = Read_Char();
        if ((b == 'Y') || (b == 'y'))
            return(TRUE);
        if ((b == 'N') || (b == 'n'))
            return(FALSE);
        Write(" Enter a Y or N --> ");
    }
}

Read_Addr(pmsg, add, cnt) /* pmsg - pointer to the output message */
                          /* add - pointer to the address */
                          /* cnt - number of bytes in the address */
    char    *pmsg, add[], cnt;
{
    char    i, j;

    for ( ; ; ) {
        Write(pmsg);
        Read(&cbuf[0], 80, &actual);
        for (j = skip(&cbuf[0]), i = 0; i < 2*cnt ; i++, j++) {
            if (('0' <= cbuf[j]) && (cbuf[j] <= '9'))
                cbuf[i] = cbuf[j] - '0';
            else
                if (('A' <= cbuf[j]) && (cbuf[j] <= 'F'))
                    cbuf[i] = cbuf[j] - 0x37;
            else
                if (('a' <= cbuf[j]) && (cbuf[j] <= 'f'))
                    cbuf[i] = cbuf[j] - 0x57;
            else {
                Write(" Illegal Character\n");
                break;
            }
        }
        if (i >= 2*cnt - 1)
            break;
    }
    for (i = 0; i <= cnt - 1; i++)
        add[(cnt - 1) - i] = cbuf[2*i] << 4 | cbuf[2*i + 1];
}

Write_Addr(padd, cnt)
    char    padd[], cnt;
{
    unsigned char   i, c[3];

    for ( ; cnt >0 ; cnt--) {
```

231421-53

/PCD/USR/CHUCK/CSRC/UAP.C

```
            i = padd[cnt-1];
            Char_To_Ascii(i, &c[0]);
            Write(&c[0]);
        }
        c[0] = '\n';
        c[1] = '\0';
        Write(&c[0]);
}

Recv_Data_1(pfd)      /* Receives the frame from the 802.2 module */

        struct  FD                  *pfd;
{
        struct FRAME_STRUCT         *prfs, *ptfs;
        struct TBD                  *ptbd, *begin_ptbd, *q;
        struct RBD                  *prbd;
        char                        *prbuf;
        int                         cnt;

        prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset);
        prfs  = (struct FRAME_STRUCT *) Build_Ptr(prbd->buff_ptr);

        switch (prfs->cmd & ~P_F_BIT)    {
            case     UI:
                if (monitor_flag)
                    break;   /* Don't put data in fifo unless in terminal mode */
                prbuf = (char *) prfs;
                prbuf += 3;  /* skip over the header info and point to the data */
                cnt = 3;
                pfd->length -= 3;
                for (; prbd != pNULL; cnt = 0, prbuf = (char *) prbd->buff_ptr){
                    for ( ; cnt < (prbd->act_cnt & 0x03FFF) && pfd->length > 0;
                                                cnt++, prbuf++, pfd->length--) {
                        while(r_buf_stat == FULL);
                        Fifo_R_In(*prbuf);
                    }
                    prbd = Build_Ptr(prbd->link);
#ifdef DEBUG
                    if (pfd->length == 0 && prbd != pNULL)
                        Fatal("Uap: Recv_Data_1(pfd) ");
#endif /* DEBUG */
                }
                break;

            case     XID:

                while ((ptbd = Get_Tbd()) == pNULL);
                ptbd->act_cnt = EOFBIT | XID_LENGTH;
                bcopy ((char *) ptbd->buff_ptr, &xid_frame[0], XID_LENGTH);
                ptfs = (struct FRAME_STRUCT *) ptbd->buff_ptr;
                ptfs->cmd = prfs->cmd;

                ptfs->dsap = prfs->ssap | C_R_BIT;  /* return the frame
                                                       to the sender */
                ptfs->ssap = ssap;
                while(!Send_Frame(ptbd, Build_Ptr(pfd->src_addr)));
```

231421-54

/PCO/USR/CHUCK/CSRC/UAP.C

```
                break;

            case    TEST:

                for (prbd = (struct RBD *) Build_Ptr(pfd->rbd_offset),
                                q = begin_ptbd = pNULL; prbd != pNULL;
                                        prbd = Build_Ptr(prbd->link)) {
                    while ((ptbd = Get_Tbd()) == pNULL);
                    if (q != pNULL)
                        q->link = Offset(ptbd);
                    else
                        begin_ptbd = ptbd;
                    ptbd->act_cnt = prbd->act_cnt;
                    bcopy((char *) ptbd->buff_ptr, (char *) prbd->buff_ptr,
                                                    ptbd->act_cnt & 0x3FFF);
                    q = ptbd;
                }

                ptfs = (struct FRAME_STRUCT *) begin_ptbd->buff_ptr;
                ptfs->cmd = prfs->cmd;

                ptfs->dsap = prfs->ssap | C_R_BIT;  /* return the frame to
                                                       the sender */
                ptfs->ssap = ssap;
                while(!Send_Frame(begin_ptbd, Build_Ptr(pfd->src_addr)));
                break;

        }
        Put_Free_RFA(pfd);    /* return the frame */
}


Fifo_T_Out()     /* called by main program */
{
        char    c;

        c = fifo_t[out_fifo_t++];

        Disable_Uart_Int();
        if (out_fifo_t == in_fifo_t)     /* if the fifo is empty */
            t_buf_stat = EMPTY; /* stop filling Transmit Buffer Descriptors */
        else        /* if the fifo was full and is now draining */
            if (t_buf_stat == FULL && out_fifo_t - 80 == in_fifo_t) { /* turn on
                                                                  the spigot */
                RTS_ONB;
                t_buf_stat = INUSE;
            }
        Enable_Uart_Int();
        return(c);
}

Fifo_T_In(c)     /* called by Uart receive interrupt */
        char    c;
{

        fifo_t[in_fifo_t++] = c;
        if (t_buf_stat == EMPTY)
```

231421-55

```
/PCO/USR/CHUCK/CSRC/UAP.C

            t_buf_stat = INUSE;  /* start filling Transmit Buffer Descriptor */
        else            /* if there are only 20 locations left, turn off the spigot */
            if (t_buf_stat == INUSE && in_fifo_t + 20 == out_fifo_t) {
                RTS_OFFB;
                t_buf_stat = FULL;
            }
}

Fifo_R_Out()    /* called by transmit interrupt */
{
    char    c;

    c = fifo_r[out_fifo_r++];

    if (out_fifo_r == in_fifo_r)     /* if the fifo is empty */
        r_buf_stat = EMPTY;
    else            /* if the fifo was full and is now draining */
        if (r_buf_stat == FULL && out_fifo_r - 81 == in_fifo_r)
            r_buf_stat = INUSE;
    return(c);
}

Fifo_R_In(c)    /* called by Recv_Data_1() */
    char    c;
{
    fifo_r[in_fifo_r++] = c;
    Disable_Uart_Int();
    if (r_buf_stat == EMPTY) {
        UART_TX_EI_B;
        Co(0);                  /* prime the interrupt */
        r_buf_stat = INUSE;
    }
    else            /* if the buffer is full, indicate it */
        if (r_buf_stat == INUSE && in_fifo_r == out_fifo_r)
            r_buf_stat = FULL;
    Enable_Uart_Int();
}

Isr_Uart()
{
    int     stat;
    char    c;

    outb(CH_B_CTL, 2);   /* point to RR2 in 8274 */

    switch(inb(CH_B_CTL) & 0x1C){ /* read 8274 interrupt vector and service it */

    case UART_TX_B:

        if (r_buf_stat == EMPTY) {
            UART_TX_DI_B;               /* if fifo is empty disable transmitter */
            RESET_TX_INT;
        }
        else
            outb (CH_B_DAT, Fifo_R_Out());
        break;
```

231421-56

```
/PCD/USR/CHUCK/CSRC/UAP.C

    case UART_RECV_ERR_B:

        outb(CH_B_CTL, 1);  /* point to RR1 in 8274 */
        stat = inb(CH_B_CTL);
        outb(CH_B_CTL, 0x30);
        if (stat & 0x0010)
            Write("\nParity Error Detected\n");
        if (stat & 0x0020)
            Write("\nOverrun Error Detected\n");
        if (stat & 0x0040)
            Write("\nFraming Error Detected\n");
        break;

    case UART_RECV_B:

        c = inb(CH_B_DAT);

        if (hs_stat == TRUE) {
            hs_stat = FALSE;  /* Flag to terminate High Speed Transmit mode */
            break;
        }

        if (local_echo)
            Co(c);              /* echo the char back to the terminal; could cause
                                   a transmit overrun if Tx interrupt is enabled */
        if (c == CTL_C)
            tmstat = FALSE;
        else
            Fifo_T_In(c);
        break;

    case EXT_STAT_INT_B:

        outb(CH_B_CTL, 0x10);    /* reset external status interrupts */
        break;

    case EXT_STAT_INT_A:

        outb(CH_A_CTL, 0x10);
        break;


    default:
        ;
    }
    EOI_80130_8274;
    EOI_8274;
}

Isr2()
{
    send_flag = TRUE;
    outb(0xEA, 125);
    outb(0xEA, 0x00);  /* Timer 1 interrupts every  125 sec */
    outb(0xE0, 0x62);     /* EOI 80130 */
}
```

231421-57

```
/PCO/USR/CHUCK/CSRC/UAP.C


Load_Lsap()
{
    int     Recv_Data_1();

    for(;;) {
        Read_Addr("\n\nEnter this Station's LSAP in Hex --> ", &ssap, 1);
        if (!Add_Dsap_Address(ssap, Recv_Data_1)) {
            Write("\n\nError: LSAP Address must be one of the following:\n");
            Write("\n       20H, 40H, 60H, 80H, A0H, C0H, E0H \n");
        }
        else break;
    }
}

Load_Multicast()
{

    for ( ; ; ) {
        Read_Addr("\nEnter the Multicast Address in Hex -->",
                                        &Multi_Addr[0], ADD_LEN);
        if ((Multi_Addr[0] & 0x01) == 0)
            Write("\nSorry, the LSB of the Multicast Address must be 1\n");
        else { if (!Add_Multicast_Address(&Multi_Addr[0])) {
                Write("\n\nSorry, Multicast Address Table is full!\n");
                break;
            }
            else {
                Write("\n\nWould you like to add another Multicast Address?");
                Write(" (Y or N) --> ");
                if (!Yes())
                    break;
            }
        }
    }
}

Remove_Multicast()
{

    for ( ; ; ) {
        Read_Addr("\nEnter the Multicast Address that you want to delete in Hex -->",
                                        &Multi_Addr[0], ADD_LEN);
        if ((Multi_Addr[0] & 0x01) == 0)
            Write("\nSorry, the LSB of the Multicast Address must be 1\n");
        else { if (!Delete_Multicast_Address(&Multi_Addr[0])) {
                Write("\n\nSorry, that Multicast Address doesn't exist!\n");
                break;
            }
            else {
                Write("\n\nWould you like to delete another Multicast Address?");
                Write(" (Y or N) --> ");
                if (!Yes())
                    break;
            }
        }
    }
}
```

231421-58

```
/PCD/USR/CHUCK/CSRC/UAP.C


Print_Addresses()
{
    struct  MAT *pmat;
    int         stat;

    Write("\n This Stations Host Address is: ");
    Write_Addr(&whoami[0], ADD_LEN);
    Write("\n The Address of the Destination Node is: ");
    Write_Addr(&Dest_Addr[0], ADD_LEN);
    Write("\n This Stations LSAP Address is: ");
    Write_Addr(&ssap, 1);
    Write("\n The Address of the Destination LSAP is: ");
    Write_Addr(&dsap, 1);
    stat = FALSE;
    for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
        if (pmat->stat == INUSE) {
            stat = TRUE;
            break;
        }
    if (stat)   {
        Write("\n The following Multicast Addresses are enabled: ");
        for (pmat = &mat[0]; pmat <= &mat[MULTI_ADDR_CNT - 1]; pmat++)
            if (pmat->stat == INUSE) {
                Write_Addr(&pmat->addr[0], ADD_LEN);
                Write("                                              ");
            }
    }
    else
        Write("\n There are no Multicast Addresses enabled.\n");
}

Init_DataLink()
{
    int     stat;

    if ((stat = Init_Llc()) == PASSED)
        Write("\n\nPassed Diagnostic Self Tests\n\n\n");
    else
        if(stat == FAILED_DIAGNOSE)
            Write("\n\nFailed: Self Test Diagnose Command\n");
    else
        if(stat == FAILED_LPBK_INTERNAL)
            Write("\n\nFailed: Internal Loopback Self Test\n");
    else
        if(stat == FAILED_LPBK_EXTERNAL)
            Write("\n\nFailed: External Loopback Self Test\n");
    else
        if(stat == FAILED_LPBK_TRANSCEIVER)
            Write("\n\nFailed: External Loopback Through Transceiver Self Test\n");
}

Init_Uap()
{

    outb(0xE0, 0x31);    /*initalize 80130 pic - ICW1 */
    outb(0xE2, 0x20);    /* ICW2 */
```

231421-59

```
/PCO/USR/CHUCK/CSRC/UAP.C

    outb(OxE2, Ox10);     /* ICW3 */
    outb(OxE2, OxOD);     /* ICW4 */
    outb(OxE2, Ox10);     /* ICW6 */
    outb(OxE2, OxFF);     /* mask all interrupts */

    outw(OxFF20, Ox0020);     /* set 80186 vector base */

    /* Initialize the 80130 timers for Terminal Mode */

    outb(OxEE, Ox34);
    outb(OxE8, OxB8);
    outb(OxE8, OxOB); /* SYSTICK set for 1 msec */
    outb(OxEE, Ox70);
    outb(OxEA, 125);
    outb(OxEA, OxOO); /* Timer 1 interrupts every  125 sec */

    /* Initialize the 8274 */
    outb(CH_B_CTL, Ox10); outb(CH_B_CTL, Ox28); outb(CH_B_CTL, Ox30);
    outb(CH_A_CTL, Ox38);
    outb(CH_B_CTL, 2); outb(CH_B_CTL, wrb[2] = Ox14);
    outb(CH_B_CTL, 1); outb(CH_B_CTL, wrb[1] = Ox15);
    outb(CH_B_CTL, 5); outb(CH_B_CTL, wrb[5] = OxEA);

    Write("\n\n\n\n\n\n\n\n\n\n\n\n\n");
    Write("                 *******************************************************\n");
    Write("                 * 82586 IEEE 802.2/802.3 Compatible Data Link Driver *\n");
    Write("                 *******************************************************\n");
    Write("\n\n\n\n\n\n\n\n");

    Init_DataLink();

    dhex = FALSE;
    monitor_flag = TRUE;

    Read_Addr("\n\nEnter the Address of the Destination Node in Hex --> ",
                                            &Dest_Addr[0], ADD_LEN);

    Load_Lsap();

    Read_Addr("\n\nEnter the Destination Node's LSAP in Hex --> ", &dsap, 1);

    Write("\n\nDo you want to Load any Multicast Addresses? (Y or N) -->");

    if (Yes())
        Load_Multicast();

    Print_Addresses();
}

Terminal_Mode()
{
    int       frame_cnt, buf_cnt;
    struct    TBD       *ptbd, *q, *begin_ptbd;
    char                *pbuf, c;

    Write("\n Would you like the local echo on? (Y or N)-->");

    if(Yes())
```

231421-60

/PCO/USR/CHUCK/CSRC/UAP.C

```
        local_echo = TRUE;
    else
        local_echo = FALSE;

    Write("\n This program will now enter the terminal mode.\n\n");
    Write("\n Press ^C then CR to return back to the menu\n\n");

    /* Initialize Fifo variables */

    out_fifo_t = in_fifo_t = out_fifo_r = in_fifo_r = 0;
    t_buf_stat = EMPTY;   r_buf_stat = EMPTY;

    EOI_80130_8274;
    Enable_Uart_Int();
    Enable_Timer_Int();
    monitor_flag = FALSE;
    tmstat = TRUE;
    while (tmstat)  {

        for (frame_cnt = 0; frame_cnt < MAX_FRAME_SIZE;   q = ptbd) {

            while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from the
                                                    data link */
            pbuf = (char *) ptbd->buff_ptr;   /* point to the buffer */
            buf_cnt = 0;

            if (frame_cnt == 0) { /* if this is the first buffer, add on IEEE 802.2
                                     header information */
                begin_ptbd = ptbd;
                *pbuf++ = dsap;
                *pbuf++ = ssap;
                *pbuf++ = UI;
                buf_cnt = 3;
            }
            else  q->link = Offset(ptbd); /* if this isn't the first buffer
                                 link the previous buffer with the new one */
            /* fill up a data link xmit buffer from async transmit fifo */
            for ( ; buf_cnt < TBUF_SIZE && frame_cnt < MAX_FRAME_SIZE;
                                      buf_cnt++, pbuf++, frame_cnt++) {
                if (frame_cnt != 0 && send_flag)
                    break;

                while (t_buf_stat == EMPTY);     /* wait until fifo has data */
                if ((c = *pbuf = Fifo_T_Out()) == CR) {
                    ++buf_cnt ; ++pbuf; ++frame_cnt;
                    break;
                }
            }
            if (c == CR || buf_cnt < TBUF_SIZE || send_flag) { /* last buffer in list */
                ptbd->act_cnt = buf_cnt | EOFBIT;
                send_flag = FALSE;
                break;
            }
        }
        while(!Send_Frame(begin_ptbd, &Dest_Addr[0])); /* keep trying until
                                                         successful */
    }
```

231421-61

/PCO/USR/CHUCK/CSRC/UAP.C

```
    Disable_Uart_Int();
    Disable_Timer_Int();
    monitor_flag = TRUE;
}

struct  TBD      *Build_Frame(cnt)
    u_short      cnt;
{
    u_short              buf_cnt, frame_cnt, i;
    struct  TBD          *ptbd, *q, *begin_ptbd;
    char                 *pbuf;

    i = 0x20; frame_cnt = 0;

    for ( ; ; q = ptbd) {
        while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from the
                                                data link */

        pbuf = (char *) ptbd->buff_ptr;   /* point to the buffer */
        buf_cnt = 0;

        if (frame_cnt == 0) { /* if this is the first buffer, add on IEEE 802.2
                                header information */
            begin_ptbd = ptbd;
            *pbuf++ = dsap;
            *pbuf++ = ssap;
            *pbuf++ = UI;
            buf_cnt = 3;
        }
        else   q->link = Offset(ptbd); /* if this isn't the first buffer
                           link the previous buffer with the new one */
        /* fill up a data link xmit buffer with ASCII characters */
        for (; buf_cnt < TBUF_SIZE && cnt > 0;
                           i++, buf_cnt++, pbuf++, cnt--, frame_cnt++) {
            *pbuf = i;
            if (i > 0x7E)
                i = 0x1F;
        }
        if (cnt == 0) { /* last buffer in list */
            ptbd->act_cnt = buf_cnt | EOFBIT;
            break;
        }
    }
    return(begin_ptbd);
}

Monitor_Mode()
{
    u_short      xmit, cnt, i;
    struct  TBD      *Build_Frame(), *ptbd;

    Write(" Do you want this station to transmit? (Y or N) --> ");
    if (Yes())
```

231421-62

/PCO/USR/CHUCK/CSRC/UAP.C

```
        for (xmit = FALSE; xmit == FALSE; ) {
        Write("\n Enter the number of data bytes in the frame --> ");
        cnt = Read_Int();
        if (cnt > 2045)
            Write ("\n Sorry, the number has to be less than 2046!\n");
        else
            xmit = TRUE;
        }

    else xmit = FALSE;

    Write("\n Hit any key to exit Monitor Mode.\n\n");

Write("  # of Good      # of Good     CRC      Alignment     No           Receive\n");
Write("   Frames          Frames     Errors     Errors      Resource      Overrun\n");
Write(" Transmitted      Received                           Errors        Errors\n");

   /* "0123456789012345678901234567890123456789012345678901234567890123456789012345678
         xxxxxxxxxx      xxxxxxxxxx     xxxx        xxxx        xxxx        xxxx
           xxxxxxxx        xxxxxxxx
                 11               25       33          44             57             71 */

    for (i = 0; i < 79; i++)
        line[i] = 0x20;
    line[79] = CR;
    line[80] = '\0';

    while ((inb(CH_B_CTL) & 1) == 0) {
        for (i = 0; i < 72; i++)
            line[i] = 0x20;
        Write_Long_Int(good_xmit_cnt, 11);
        Write_Long_Int(recv_frame_cnt, 25);
        Write_Short_Int(scb.crc_errs, 33);
        Write_Short_Int(scb.aln_errs, 44);
        Write_Short_Int(scb.rsc_errs, 57);
        Write_Short_Int(scb.ovr_errs, 71);
        Write(&line[0]);
        if (xmit) {
            ptbd = Build_Frame(cnt);
            while(!Send_Frame(ptbd, &Dest_Addr[0]));
        }
    }
    i = Ci();
}

Hs_Xmit_Mode()
{
    struct  TBD  *ptbd;

    Write("\n Hit any key to exit High Speed Transmit Mode.\n\n");

    hs_stat = TRUE;
    EOI_80130_8274;
    Enable_Uart_Int();

    /* Execute this loop until a recv char interrupt happends at Uart */
```

231421-63

/PCO/USR/CHUCK/CSRC/UAP.C

```
    while (hs_stat) {
        while ((ptbd = Get_Tbd()) == pNULL); /* get a xmit buffer from
                                                the data link */
        ptbd->act_cnt |= EOFBIT;            /* set the End Of Frame bit */
        while(!Send_Frame(ptbd, &Dest_Addr[0])); /* Send Frame */
    }

    Disable_Uart_Int();
}

Print_Cnt()
{
    char     ch[11], base, dwidth, width, i;
    unsigned     long     temp;

      (dhex) {
        dwidth = 3;
        width = 4;
        base = 16;
    }
    else {
        base = 10;
        dwidth = 10;
        width = 5;
    }

    Write("\n\n Good frames transmitted: ");
    for (i = 1; i <= 11 - dwidth; i++)
        Co(SP);
    Int_To_Ascii(good_xmit_cnt, base, ' ', &ch[0], dwidth);
    for (i = dwidth - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Good frames received: ");
    for (i = 1; i <= 15 - dwidth; i++)
        Co(SP);
    Int_To_Ascii(recv_frame_cnt, base, ' ', &ch[0], dwidth);
    for (i = dwidth - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n CRC errors received: ");
    for (i = 1; i <= 15 - width; i++)
        Co(SP);
    temp = scb.crc_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("  Alignment errors received: ");
    for (i = 1; i <= 10 - width; i++)
        Co(SP);
    temp = scb.aln_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
    for (i = width - 1; i >= 0; i--)
        Co(ch[i]);
    Write("\n\n Out of Resource frames: ");
    for (i = 1; i <= 12 - width; i++)
        Co(SP);
    temp = scb.rsc_errs;
    Int_To_Ascii(temp, base, ' ', &ch[0], width);
```

231421-64

/PCD/USR/CHUCK/CBRC/UAP.C

```
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("  Receiver overrun frames: ");
     for (i = 1; i <= 12 - width; i++)
         Co(SP);
     temp = scb.ovr_errs;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("\n\n 82586 Reset: ");
     for (i = 1; i <= 23 - width; i++)
         Co(SP);
     temp = reset_cnt;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("  Transmit underrun frames: ");
     for (i = 1; i <= 11 - width; i++)
         Co(SP);
     temp = underrun_cnt;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("\n\n Lost CRS: ");
     for (i = 1; i <= 26 - width; i++)
         Co(SP);
     temp = no_crs_cnt;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("  SQE errors: ");
     for (i = 1; i <= 25 - width; i++)
         Co(SP);
     temp = sqe_err_cnt;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("\n\n Maximum retry: ");
     for (i = 1; i <= 21 - width; i++)
         Co(SP);
     temp = max_col_cnt;
     Int_To_Ascii(temp, base, ' ', &ch[0], width);
     for (i = width - 1; i >= 0; i--)
         Co(ch[i]);
     Write("  Frames that deferred: ");
     for (i = 1; i <= 15 - dwidth; i++)
         Co(SP);
     Int_To_Ascii(defer_cnt, base, ' ', &ch[0], dwidth);
     for (i = dwidth - 1; i >= 0; i--)
         Co(ch[i]);
}


Print_Help()
{
     Write ("\n\n Commands are:\n\n");
     Write (" T - Terminal Mode                              M - Monitor Mode\n");
```

231421-65

/PCO/USR/CHUCK/CSRC/UAP.C

```
    Write (" X - High Speed Transmit Mode            V - Change Transmit Statistics\n");
    Write (" P - Print All Counters                  C - Clear All Counters\n");
    Write (" A - Add a Multicast Address             Z - Delete a Multicast Address\n");
    Write (" S - Change the SSAP Address             D - Change the DSAP Address\n");
    Write (" N - Change Destination Node Address     L - Print All Addresses\n");
    Write (" R - Re-Initialize the Data Link         B - Change the number Base\n");
}

Main()
{
    int     c;

    Init_Uap();
    Print_Help();

    for (;;) {
        Write ("\n\n Enter a command, type H for Help --> ");
        c = Read_Char();
        switch (Lower_Case(c)) {

        case 'h':
            Print_Help();
            break;
        case 'm':
            Monitor_Mode();
            break;
        case 't':
            Terminal_Mode();
            break;
        case 'x':
            Hs_Xmit_Mode();
            break;
        case 'v':
            Write("\n Transmit Statistics are now ");
            if (flags.stat_on == 1)
                Write("on.\n Would you like to change it ? (Y or N) --> ");
            else
                Write("off.\n Would you like to change it ? (Y or N) --> ");
            if (Yes()) {
                if (flags.stat_on == 1)
                    flags.stat_on = 0;
                else flags.stat_on = 1;
            }
            break;
        case 'p':
            Print_Cnt();
            break;
        case 'c':
            Clear_Cnt();
            break;
        case 'a':
            Load_Multicast();
            break;
        case 'z':
            Remove_Multicast();
            break;
        case 's':
```

231421-66

/PCO/USR/CHUCK/CSRC/UAP.C

```
            Delete_Dsap_Address(ssap);
            Load_Lsap();
            break;
        case 'd':
            Read_Addr("\n\nEnter the Destination Node's LSAP in Hex --> ", &dsap, 1);
            break;
        case 'n':
            Read_Addr("\n\nEnter the Address of the Destination Node in Hex --> ",
                                                        &Dest_Addr[0], ADD_LEN);
            break;
        case 'l':
            Print_Addresses();
            break;
        case 'r':
            Software_Reset();
            Init_DataLink();
            Add_Dsap_Address(ssap, Recv_Data_1);
            break;
        case 'b':
            Write("\n The current base is ");
            if (dhex == TRUE)
                Write("Hex.\n Would you like to change it ? (Y or N) --> ");
            else
                Write("Decimal.\n Would you like to change it ? (Y or N) --> ");
            if (Yes()) {
                if (dhex == TRUE)
                    dhex = FALSE;
                else dhex = TRUE;
            }
            break;

        default:
            Write ("\n Unknown command\n");
            break;
        }
    }
}
```

231421-67

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

name     c_assy_support

stack    segment stack    'stack'
stktop   label   word
stack    ends

DLD_DATA                  segment public  'DATA'
extrn    SEGMT_:word                      ; data segment address
DLD_DATA                  ends

UAP_DATA          segment public    'DATA'
UAP_DATA          ends

DLD_CODE          segment public  'CODE'
        extrn      Isr_Timeout_:far, Isr_586_:far, Isr7_:far
        extrn      Isr6_:far, Isr5_:far, Isr1_:far
DLD_CODE          ends

UAP_CODE          segment public  'CODE'
        extrn     Isr_Uart_:far, Isr2_:far, Main_:far
UAP_CODE          ends


DQ_CODE           segment public  'CODE'

        public     inw_, outw_, init_intv_, enable_, disable_, Build_Ptr_
        public     Offset_, begin, inb_, outb_

arg1    equu      [BP + 6]
arg2    equ       [BP + 8]

        assume    CS:DQ_CODE
        assume    DS:DLD_DATA

;+
;       initialization program for the 82586 data link driver
;-

begin:

        sti
        mov       ax, DLD_DATA ;get base of dgroup and
        mov       SEGMT_, ax        ;pass the segment value to the c program
        mov       ds, ax
        call Main_                  ;go to the c program
        hlt


inb_    proc      far
        push      BP
        mov       BP, SP
        push      DX
        mov       DX, arg1
        in        AL, DX
        pop       DX
        mov       SP, BP
```

231421-68

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
        pop     BP
        ret
inb_    endp

outb_   proc    far
        push    BP
        mov     BP, SP
        push    DX
        push    AX
        mov     DX, arg1
        mov     AX, arg2
        out     DX, AL
        pop     AX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
outb_   endp

inw_    proc    far
        push    BP
        mov     BP, SP
        push    DX
        mov     DX, arg1
        in      AX, DX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
inw_    endp

outw_   proc    far
        push    BP
        mov     BP, SP
        push    DX
        push    AX
        mov     DX, arg1
        mov     AX, arg2
        out     DX, AX
        pop     AX
        pop     DX
        mov     SP, BP
        pop     BP
        ret
outw_   endp

Build_Ptr_      proc    far
        push    BP
        mov     BP, SP
        mov     DX, DLD_DATA
        mov     AX, arg1
        mov     SP, BP
        pop     BP
        ret
Build_Ptr_      endp

Offset_ proc    far
```

231421-69

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
        push    BP
        mov     BP, SP
        mov     AX, arg1
        mov     SP, BP
        pop     BP
        ret
Offset_ endp

serve_int_isr   proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_586_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_isr   endp

serve_int_8274  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, UAP_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_Uart_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
```

231421-70

/PCD/USR/CHUCK/CSRC/ASSY.ASM

```
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_8274  endp

serve_int_timeout       proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr_Timeout_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int_timeout       endp

serve_int7_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr7_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
```

231421-71

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
        iret
serve_int7_isr  endp

serve_int6_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr6_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int6_isr  endp

serve_int5_isr  proc    far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr5_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int5_isr  endp
```

231421-72

/PCO/USR/CHUCK/CSRC/ASSY.ASM

```
serve_int2_isr  proc      far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, UAP_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr2_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int2_isr  endp

serve_int1_isr  proc      far
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES

        mov     AX, DLD_DATA
        mov     DS, AX
        mov     ES, AX

        call    Isr1_

        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        iret
serve_int1_isr  endp


enable_ proc      far
        sti
```

231421-73

```
/PCO/USR/CHUCK/CSRC/ASSY.ASM

        ret
enable_ endp

disable_        proc    far
        cli
        ret
disable_        endp

init_intv_      proc    far
        push    DS
        push    AX

        xor     AX, AX
        mov     DS, AX

        ; Interrupt types for the 186/51 COMMputer

        mov     DS:word ptr  80h, offset serve_int_8274        ; int 0
        mov     DS:word ptr  82h, DG_CODE
        mov     DS:word ptr  84h, offset serve_int1_isr        ; int 1
        mov     DS:word ptr  86h, DG_CODE
        mov     DS:word ptr  88h, offset serve_int2_isr        ; int 2
        mov     DS:word ptr  8Ah, DG_CODE
        mov     DS:word ptr  8Ch, offset serve_int_isr         ; int 3
        mov     DS:word ptr  8Eh, DG_CODE
        mov     DS:word ptr  90h, offset serve_int_timeout     ; int 4
        mov     DS:word ptr  92h, DG_CODE
        mov     DS:word ptr  94h, offset serve_int5_isr        ; int 5
        mov     DS:word ptr  96h, DG_CODE
        mov     DS:word ptr  98h, offset serve_int6_isr        ; int 6
        mov     DS:word ptr  9Ah, DG_CODE
        mov     DS:word ptr  9Ch, offset serve_int7_isr        ; int 7
        mov     DS:word ptr  9Eh, DG_CODE


        pop     AX
        pop     DS
        ret

init_intv_      endp

DG_CODE ends
        end             begin, ds:dld_data, ss:stack:stktop
```

231421-74

intel®

APPLICATION NOTE

AP-274

November 1986

Implementing Ethernet/Cheapernet with the Intel 82586

KIYOSHI NISHIDE
APPLICATIONS ENGINEER

Order Number: 292010-002

PREFACE

Intel's three VLSI chip set, the 82586, 82501, and 82502, is a complete solution for IEEE 802.3 10M bps LAN standards—10BASE5 (Ethernet) and 10BASE2 (Cheapernet). The 82586 is an intelligent peripheral which completely manages the processes of transmitting and receiving frames over a network under the CSMS/CD protocol. The 82586 with its on-chip four DMA channels offloads the host CPU of the tasks related to managing communication activities. The chip, for example, does not depend on the host CPU for time critical functions, such as transmissions/retransmissions and receptions of frames. The 82501 is a 10 MHz serial interface chip specially designed for the 82586. The primary function of the 82501 is to perform Manchester encoding/decoding, provide 10 MHz transmit and receive clocks to the 82586, and drive the transceiver (AUI) cable in Ethernet applications. In addition, the 82501 provides a loopback function and on-chip watchdog timer. The 82502 is a CMOS transceiver chip. The 82502 is the chip which actually drives the coaxial cable used for Ethernet or Cheapernet.

This Ap Note presents a design example of a simple but general Ethernet/Cheapernet board based on the three chip set. The board is called LANHIB (LAN High Integration Board) and uses an 80186 microprocessor as the host CPU. The LANHIB is an independent single board computer and requires only a power supply and ASCII terminal. Demo software, called TSMS (Traffic Simulator and Monitor Station) is also included in this Ap Note. The TSMS program is a network debugger and exercise tool used to exercise the 82586. In addition, flowcharts for troubleshooting are provided in order to minimize debugging time of the LANHIB board.

1.0 INTRODUCTION

A brief overview of the CSMA/CD protocol is described in Section 2. Ethernet and Cheapernet are also compared in this section.

Section 3 discusses hardware of the LANHIB in detail. This section should be helpful not only to understand the LANHIB, but also to learn in general how a system based on the three chip set can be put together. Since the 82502 involves analog circuitry, an explanation on proper layout is provided.

Demo software is presented in Section 4.0. It covers EPROM programming procedures and three sample sessions. Step by step operations at a terminal are illustrated in the figures.

Section 5 describes LANHIB troubleshooting procedures. Flowcharts are used to guide troubleshooting.

Complete LANHIB schematics and parts list are found in Appendix A. If a LANHIB is to be built, the schematics and Section 5 can be submitted to an available wire wrap facility. In parallel to board construction, Sections 3 and 4 can be studied. A factory wire wrap board for the LANHIB is offered at a discount price by Augat Corporation. Please return the enclosed card for more information.

Listing of the TSMS program and LANHIB Initialization Routine are in Appendix B. The source codes and related files are available on a diskette by returning the card enclosed in this design kit or through Insite (Intel's Software Index and Technology Exchange Library).

2.0 ETHERNET/CHEAPERNET OVERVIEW

2.1 CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a simple and efficient means of determining how a station transmits information over common medium that is shared with other stations. CSMA/CD is the access method defined by the IEEE 802.3 standard.

Carrier Sense (CS) means that any station wishing to transmit "listens" first. When the channel is busy (i.e., some other station is transmitting) the station waits (defers) until the channel is clear before transmitting ("listen before talk").

Multiple Access (MA) means that any stations wishing to transmit can do so. No central controller is needed to decide who is able to transmit and in what order.

Collision Detection (CD) means that when the channel is idle (no other station is transmitting) a station can start transmitting. It is, however, possible for two or more stations to start transmitting simultaneously causing a "collision". In the event of a collision, the transmitting stations will continue transmitting for a fixed time to ensure that all transmitting stations detect the collision. This is known as jamming. After the jam, the stations stop transmitting and wait a random period of time before retrying. The range of random wait times increases with the number of successive collisions such that collisions can be resolved even if a large number of stations are colliding.

There are three significant advantages to the CSMA/CD protocol. The first and foremost is that CSMA/CD is a proven technology. One CSMA/CD network, Ethernet, has been used by Xerox since 1975. Ethernet is so well understood and accepted that IEEE adopted

it (with minor changes) as the IEEE 802.3 10Base5 (10 Mbps, Baseband, 500 meters per segment) standard. Reliability is the second advantage to the 802.3 protocol. This media access method enables the network to operate without central control or switching. Thus, if a single station malfunctions, the rest of the network can continue operation. Finally, since CSMA/CD networks are passive and distributed in nature, they allow for easy expansion. New nodes can be added at any time without reinitializing the entire network.

2.2 Ethernet and Cheapernet

The IEEE 802.3 Type 10BASE5 standard (Ethernet) has gained wide acceptance by both large and small corporations as a high speed (10 Mbps) Local Area Network. The Ethernet channel is a low noise, shielded 50Ω coaxial cable over which information is transmitted at 10 million bits per second. Each segment of cable can be up to 500 meters in length and can be connected to longer network lengths using repeaters. Repeaters regenerate the signal from one cable segment onto another. At each end of a cable segment a terminator is attached. This passive device provides the proper electrical termination to eliminate reflections. The transceiver transmits and receives signals on the coaxial cable. In addition, it isolates the node from the channel so that a failure within the node will not affect the rest of the network. The transceiver is also responsible for detecting collisions—simultaneous transmissions by two or more stations. Ethernet transceivers are connected to the network coaxial cable using a simple tap, and to the station it serves via the transceiver cable which can be

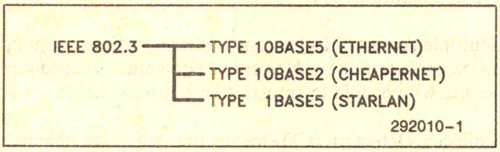


Figure 1. Different Implementations of IEEE 802.3 (Note: "10BASE5", for example, implies 10 Mbps, Baseband, and 500 meters span.)

up to 50 meters in length. The transceiver cable is made of four individually shielded twisted pairs of wires. An Ethernet interface at a computer (DTE), which includes a serial interface and data link controller, provides the connection to the user or server station. It also performs frame manipulation, addressing, detecting transmission errors, network link management, and encoding and decoding of the data to and from the transceiver.

The IEEE 802.3 Type 10BASE2 (Cheapernet) has the same functional and electrical specifications as Type 10BASE5 (Ethernet) with only two exceptions in physical (or rather mechanical) characteristics. Cheapernet is as shown in Figure 1 just a different implementation of the IEEE standard. Ethernet and Cheapernet are both 10 million bits/second CSMA/CD LANs and use the identical network parameters, such as slot time = 51.2 μs. Ethernet and Cheapernet can, therefore, be built by the same VLSI components with the same software (Figure 2).

The two physical differences attribute to the cost reduction purpose of Cheapernet—cheaper implementation of Ethernet. First, the cable used in Cheapernet may be a lower cost 50Ω coaxial cable than the one for Ethernet. The most common coaxial cable for Cheapernet is RG58 which cost about $0.15/ft. A typical Ethernet cable costs about $0.83/ft.

Second, the transceiver is integrated into the DTE in Cheapernet. The coaxial cable physically comes to the DTE, connects to the transceiver within the DTE, and goes to the next DTE (see Figure 3). The kind of connector used at the DTE is an off-the-shelf BNC "T" connector. Topology is, therefore, a simple daisy chaining. This cabling scheme contributes to further cost reduction due to omission of the Transceiver (AUI) Cable, cheaper connectors, and easier installation. The Ethernet transceiver cable costs about $1.49/ft. More flexible thin coaxial cables and familiar BNC "T" connectors are making Cheapernet a user installable Ethernet compatible network.

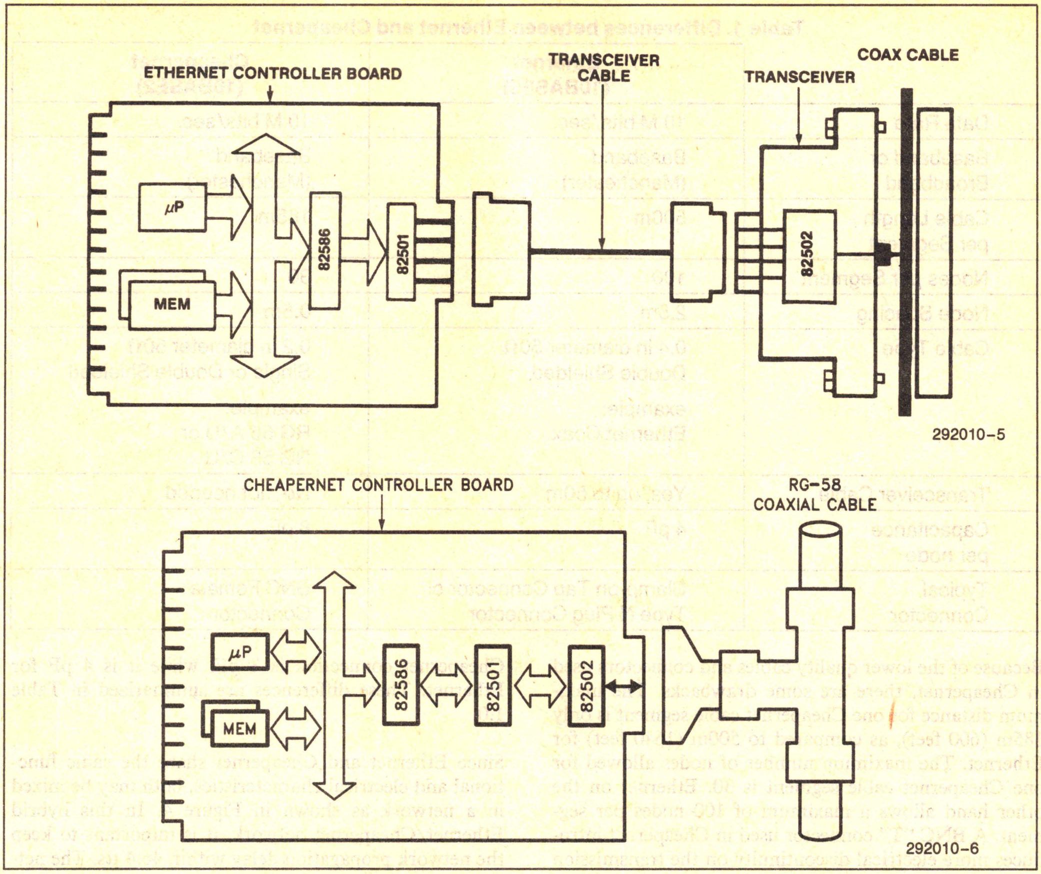


Figure 2. 82586/82501/82502 in Ethernet and Cheapernet

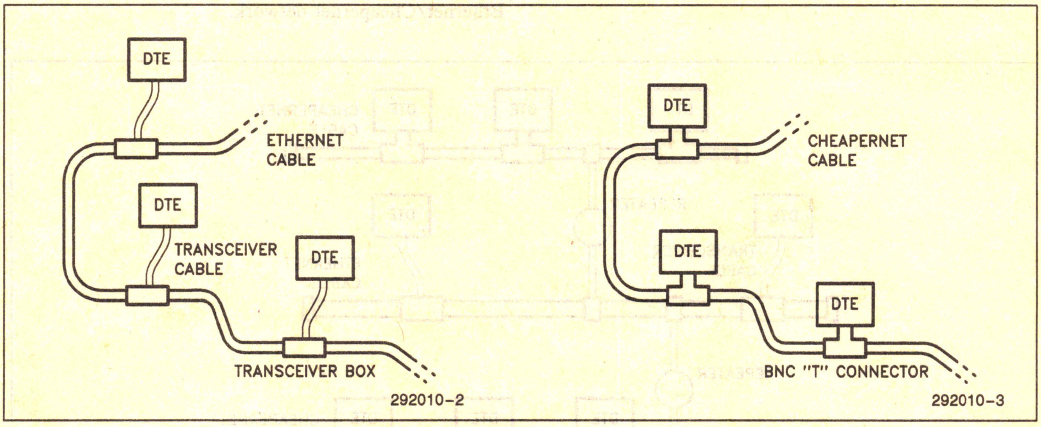


Figure 3. Ethernet Cabling vs Cheapernet Cabling

Table 1. Differences between Ethernet and Cheapernet

	Ethernet (10BASE5)	Cheapernet (10BASE2)
Data Rate	10 M bits/sec.	10 M bits/sec.
Baseband or Broadband	Baseband (Manchester)	Baseband (Manchester)
Cable Length per Segment	500m	185m
Nodes per Segment	100	30
Node Spacing	2.5m	0.5m
Cable Type	0.4 in diameter 50Ω Double Shielded example: Ethernet Coax.	0.2 in diameter 50Ω Single or Double Shielded example: RG 58 A/U or RG 58 C/U
Transceiver Cable	Yes, up to 50m	No, not needed
Capacitance per node	4 pF	8 pF
Typical Connector	Clamp-on Tap Connector or Type N Plug Connector	BNC Female Connector

Because of the lower quality cables and connectors used in Cheapernet, there are some drawbacks. The maximum distance for one Cheapernet cable segment is only 185m (600 feet), as compared to 500m (1640 feet) for Ethernet. The maximum number of nodes allowed for one Cheapernet cable segment is 30. Ethernet on the other hand allows a maximum of 100 nodes per segment. A BNC "T" connector used in Cheapernet introduces more electrical discontinuity on the transmission line than the clamp-on tap connector widely used for Ethernet. The maximum capacitance load allowed at a Cheapernet connection is 8 pF, while it is 4 pF for Ethernet. These differences are summarized in Table 1.0.

Since Ethernet and Cheapernet share the same functional and electrical characteristics, both may be mixed in a network as shown in Figure 4. In this hybrid Ethernet/Cheapernet network, it is important to keep the network propagation delay within 46.4 μs. The network may be expanded as required within this round trip propagation delay limit. Ethernet, for example, may serve as a backbone for Cheapernet in a hybrid Ethernet/Cheapernet network.

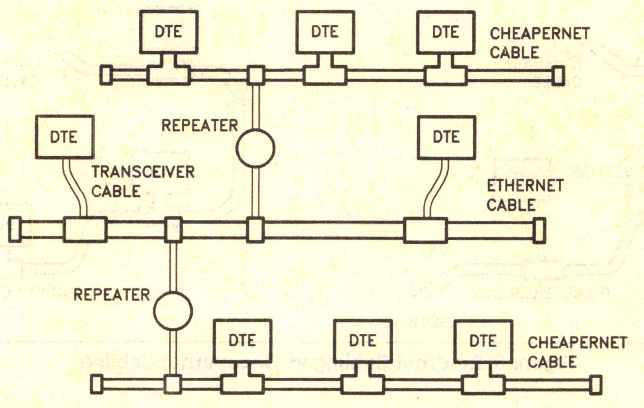


Figure 4. Ethernet/Cheapernet Hybrid Network

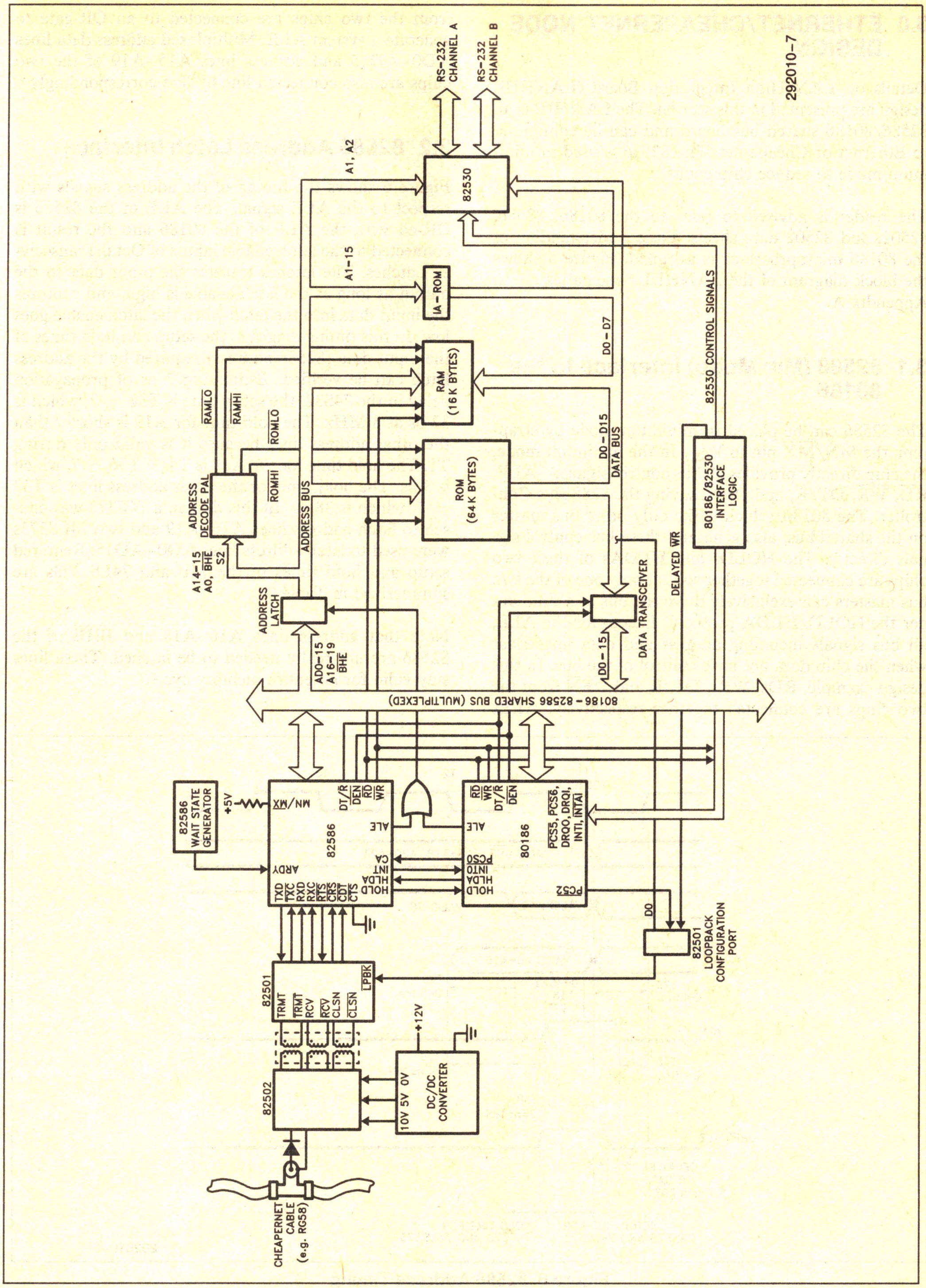


Figure 5. LANHIB Block Diagram

3.0 ETHERNET/CHEAPERNET NODE DESIGN

Details on LAN High Integration Board (LANHIB) design are presented in this section. The LANHIB is an 82586/80186 shared bus board and can be configured to Ethernet or Cheapernet. The 82586 is used in minimum mode to reduce chip count.

The reader is advised to refer to the 80186, 82586, 82501, and 82502 data sheets. Basic understanding of the 80186 microprocessor is assumed. Figure 5 shows the block diagram of the LANHIB. Schematics are in Appendix A.

3.1 82586 (Min Mode) Interface to the 80186

The 82586 can be placed in minimum mode by strapping the MN/$\overline{\text{MX}}$ pin to V_{CC}. In the minimum mode, the chip directly provides all bus control signals—ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, DT/$\overline{\text{R}}$, and $\overline{\text{DEN}}$, saving the 8288 Bus Controller. The 80186, which is the only other bus master on the shared bus, also generates these bus control signals directly. The HOLDs and HLDAs of these two chips are connected together so that only one of the two bus masters can exclusively drive the bus at a time under the HOLD/HLDA protocol. Except for the ALE, all bus signals including address and data lines float when the chip does not have control of the bus. In this design example, $\overline{\text{RD}}$s, $\overline{\text{WR}}$s, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$ from the two chips are connected together respectively. ALEs from the two chips are connected to an OR-gate to generate a system ALE. Multiplexed address data lines AD0–AD15 and address lines A15–A19 of the two chips are also connected line by line correspondingly.

3.2 82586 Address Latch Interface

Figure 6 shows the timing of the address signals with respect to the ALE signal. The ALE of the 82586 is OR-ed with the ALE of the 80186 and the result is connected to the latch enable inputs of Octal Transceiver Latches. The latches transfer the input data to the output as long as the latch enable is high, and captures the input data into the latch when the latch enable goes low. In this timing diagram, the setup and hold times of the input data (82586 address) required by the address latch can be verified. Estimating 7 ns of propagation delay in the 74S32, the setup time is T38 + 7, which is 32 ns at 8 MHz. The hold time for A19 is shorter than the other address lines because it is valid only during T1. The hold time for the A19 is T4 − T36 − 7, which is 3 ns. The hold time for the other address lines is T39 − 7, which is 38 ns. In this design, a 74F373 was chosen to latch address lines A16–A19 and two 74LS373s were used to latch address lines AD0–AD15. Required setup and hold times of the 74F and 74LS 373s are summerized in Table 2.

Note that address lines A16–A18 and $\overline{\text{BHE}}$ of the 82586 are not really needed to be latched. These lines stay valid for an entire memory cycle.

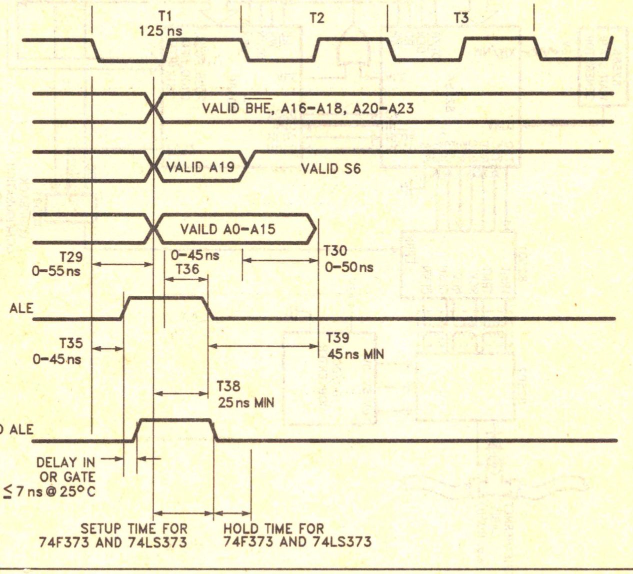


Figure 6. 82586 Address Timing

Table 2. 74F and 74LS Data Setup and Hold Time Specifications at 25°C

	74F373			74LS373			Unit
	Min	Nom	Max	Min	Nom	Max	
Data Setup Time	2↓			5↓			ns
Data Hold Time	3↓			20↓			ns

3.3 80186 Address Latch Interface

The address latch used by the 82586 is shared by the 80186. Figure 7 shows the 80186 address line timing with respect to the ALE. Again estimating 7 ns delay in the 74S32, the setup time for the latch is TAVAL + 7 and the hold time is TLLAX − 7. These are 37 ns and 23 ns respectively at 8 MHz. Comparing to the required values shown in Table 2, it is quite obvious that the setup and hold times of the latch are met by wide margins. Note that the 80186's address lines A16–A18 and $\overline{\text{BHE}}$ are not valid for an entire memory cycle; therefore, they have to be latched.

3.4 82586 Memory Interface

The 74LS373 has a delay of 18 ns for input data to reach the output assuming the latch enable is high. A demultiplexed valid address (output of the address latch), therefore, becomes available after T29 + 18 measuring from the beginning of T1 (Figure 8). The demultiplexed address remains valid until the ALE of the next memory access becomes active. Upper address lines, A14 through A20, are connected to a 16L8 PAL, which provides address decode logic for all memory devices. The PAL truth table is in Appendix A. The PAL has a maximum of 35 ns propagation delay, so chip selects will become active after 55 + 18 + 35 ns (max.) from the beginning of T1 as indicated in Figure 8. Since address decode logic is implemented by a PAL, any memory expansion would only require a reprogramming of this PAL.

Two 74LS245 bus transceiver chips are controlled by the DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$. Output enable and disable times of the 74LS245 are 40 and 25 ns respectively. The maximum propagation delay when the output enable is active is 12 ns.

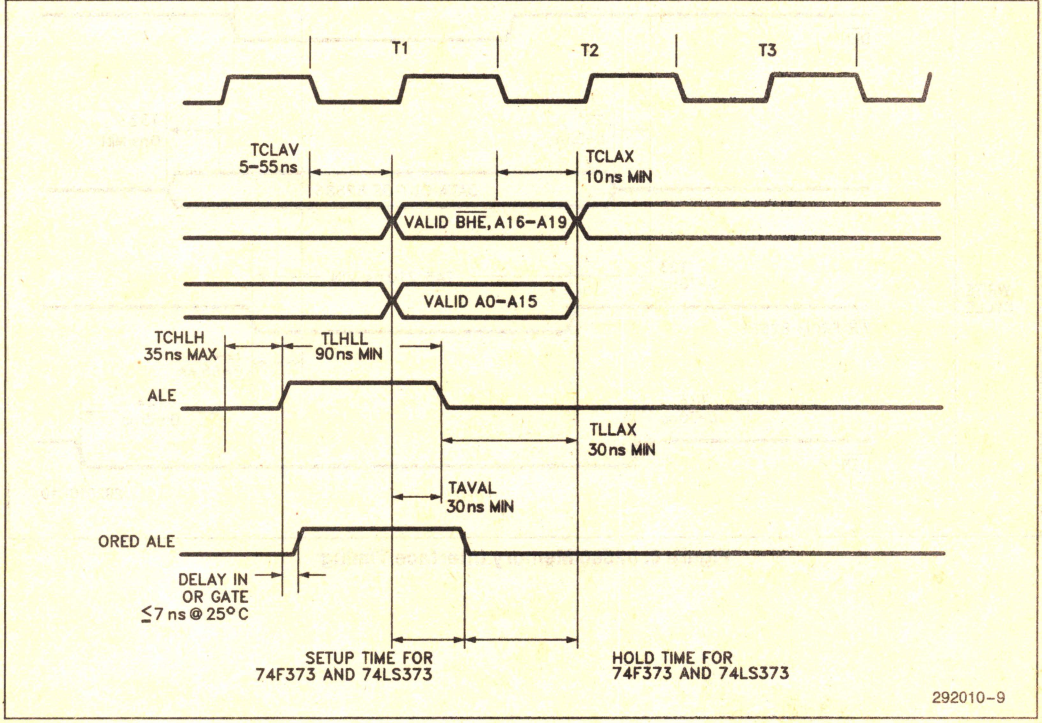


Figure 7. 80186 Address Timing

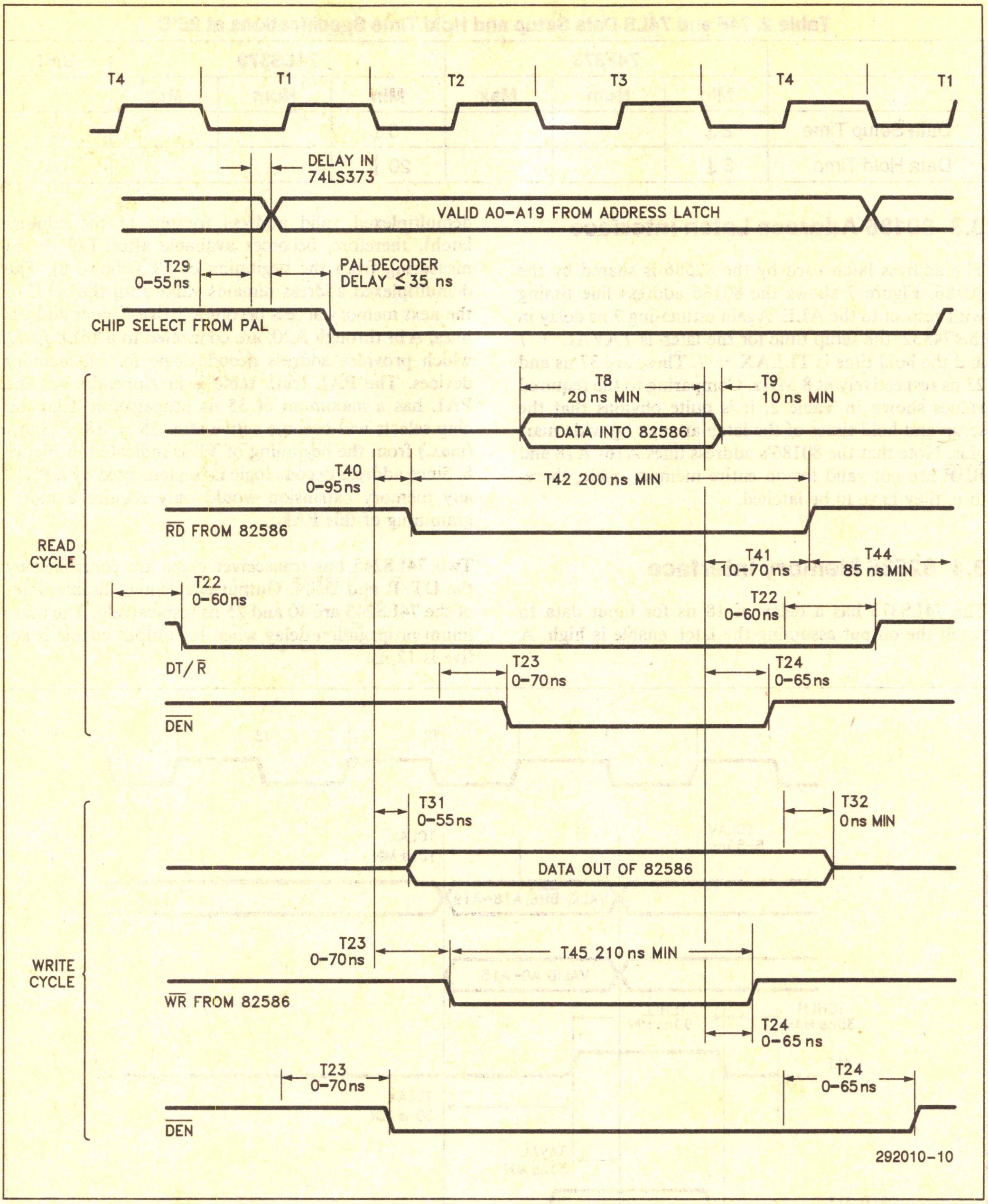


Figure 8. 82586 Memory Interface Timing

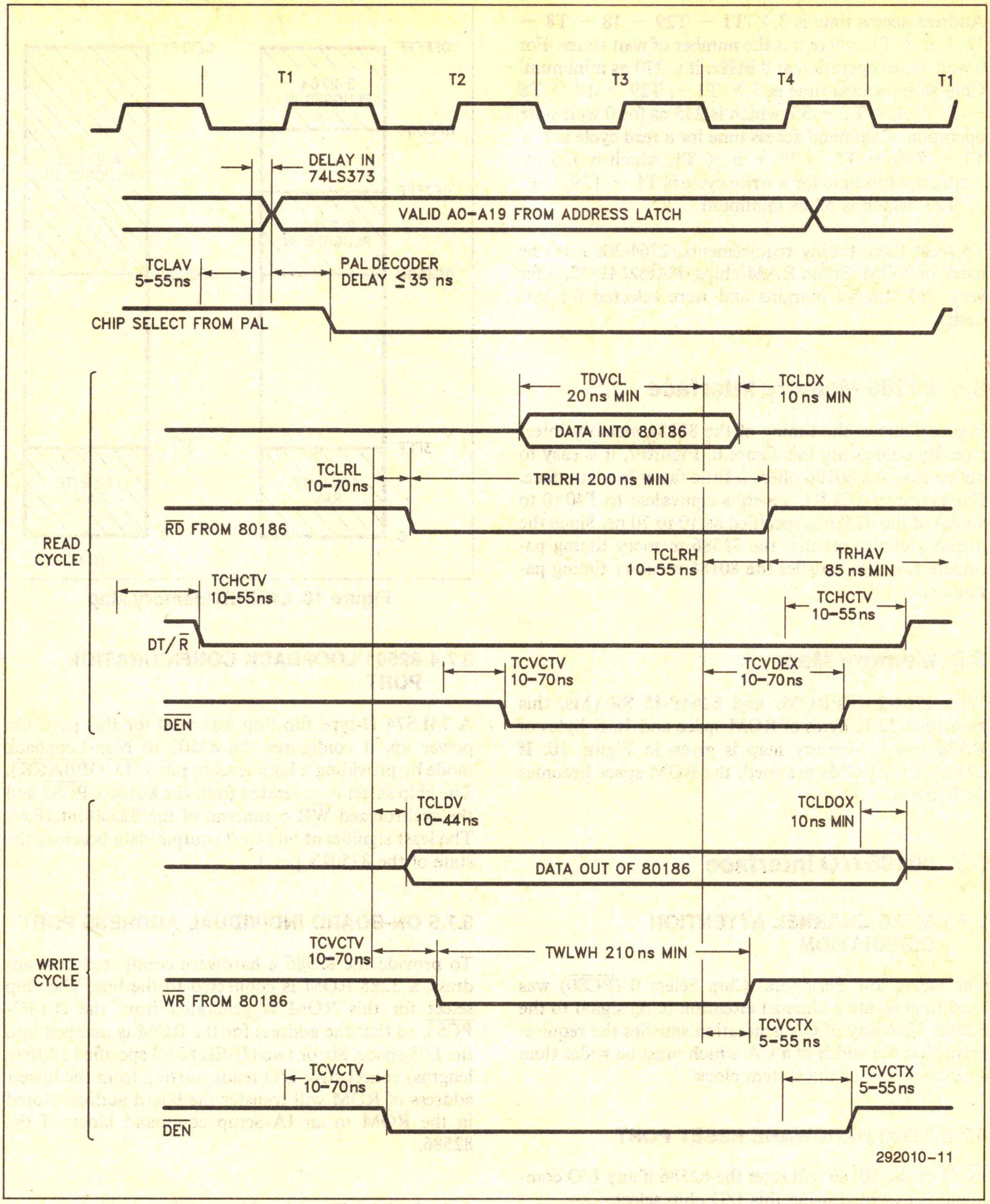


Figure 9. 80186 Memory Interface Timing

Address access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1, where n is the number of wait states. For 0 wait states operation at 8 MHz, it is 270 ns minimum. Chip select access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1 − 35, which is 235 ns for 0 wait state operation. Command access time for a read cycle is 2 × T1 − T40 − T8 − 12 + n × T1, which is 123 ns. Address setup time for a write cycle is T1 − T29 − 18 + T23, which is 52 ns minimum.

To meet these timing requirements, 2764-20s must be used for ROM. Static RAM chips, HM6264P-15, offer very wide timing margins and were selected for this design.

3.5 80186 Memory Interface

Figure 9 shows the timing of the 80186 memory interface. By comparing this figure to Figure 7, it is easy to notice that the 80186 offers a little faster bus interface. For example, TCLRL which is equivalent to T40 (0 to 95 ns) of the 82586 is specified as 10 to 70 ns. Since the memory choice satisfies the 82586 memory timing parameters, it also satisfies the 80186 memory timing parameters.

3.6 Memory Map

With 2764-20 EPROMs and 6264P-15 SRAMs, this board has 32 K bytes of ROM space and 16 K bytes of RAM space. Memory map is given in Figure 10. If 27128-20 EPROMs are used, the ROM space becomes 64 K bytes.

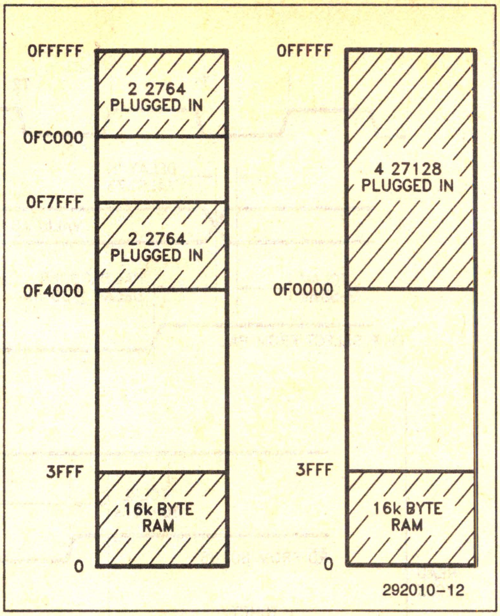


Figure 10. LANHIB Memory Map

3.7 80186 I/O Interface

3.7.1 82586 CHANNEL ATTENTION GENERATION

The active low Peripheral Chip Select 0 ($\overline{PCS0}$) was used to generate a channel attention (CA) signal to the 82586. This way of CA generation satisfies the requirement that the width of a CA which must be wider than a clock period of the system clock.

3.7.2 82586 HARDWARE RESET PORT

$\overline{PCS1}$ of the 80186 will reset the 82586 if any I/O command is executed using this I/O chip select.

3.7.3 82530 INTERFACE

82530 interface to the 80186 was derived from the design example presented in the 82530 SCC-80186 Interface Ap Brief. This document is attached to this Ap Note as Appendix C.

3.7.4 82501 LOOPBACK CONFIGURATION PORT

A 74LS74 D-type flip flop was used for this port. On power up, it configures the 82501 to Non-Loopback mode by providing a high level to pin 3 ($\overline{LOOPBACK}$). The chip select is generated from the 80186's $\overline{PCS2}$ and the sychronized $\overline{WR}$ command of the 82530 interface. The least significant bit of I/O output data becomes the state of the 82501's pin 3.

3.7.5 ON-BOARD INDIVIDUAL ADDRESS PORT

To provide the 82586 a hardware configured host address, a 32x8 ROM is connected to the bus. The chip select for this ROM is generated from the 80186's $\overline{PCS3}$, so that the address for the ROM is mapped into the I/O space. Six or two (IEEE 802.3 specified address lengths) consecutive I/O reads starting from the lowest address of ROM will transfer the board address stored in the ROM to an IA-Setup command block of the 82586.

3.8 82586 Ready Signal Generation

82586 asynchronous ready (ARDY) signal is generated from a shift register. The shift register provides the 82586 a "normally ready" signal. When a wait state is needed, the ready signal is dropped to the low state. As shown in Table 3, the 82586 can be programmed to have 0 to 8 wait states by setting the DIP switch properly. Even though the on-board memory devices are

Table 3. DIP Switch Settings for Various Numbers of 82586 Wait States

Dip Switch Setting 7	6	5	4	3	2	1	0	Number of Wait States the 82586 Inserts
1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0	2
1	1	1	1	1	0	0	0	3
1	1	1	1	0	0	0	0	4
1	1	1	0	0	0	0	0	5
1	1	0	0	0	0	0	0	6
1	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	0	8

1 = Switch Open
0 = Switch Closed

fast enough for 0 wait states operation, this programmable wait state capability was added so that the effect of wait states on the 82586 performance could be evaluated.

3.9 82501 Circuits

Since the 82501 is designed to work with the 82586, no interfacing circuits are required.

The transceiver cable side of the 82501 requires some passive components. The receive and collision differential inputs must be terminated by 78Ω ±5% resistors. Common mode voltages on these differential inputs are established internally. 240Ω ±5% pull down resistors must be connected on the TRMT and $\overline{\text{TRMT}}$ output pins.

A 0.022 μF ±10% capacitor connected between pin 1 and 2 of the 82501 is for the analog phase-locked loop.

Connected between the X1 and X2 pins is a 20 MHz parallel resonant quartz crystal (antiresonant with 20 pF load fundamental mode). An internal divide-by-two counter generates the 10 MHz clock. Since both Ethernet and Cheapernet tolerate an error of only ±0.01% in bit rate, a high quality crystal is recommended. The accuracy of a crystal should be equal to or better than ±0.002% @ 25°C and ±0.005% for 0–70°C. A 30–35 pF capacitor is connected from each crystal pin (X1 and X2) to ground in order to adjust effective capacitance load for the crystal, which should be about 20 pF including stray capacitance.

3.10 82502 Circuits

3.10.1 ISOLATION AND POWER REQUIREMENTS

The IEEE 802.3 standard requires an electrical isolation within the transceiver (MAU). Cheapernet (10BASE2) requires the isolation means to withstand 500V ac, rms for one minute. Ethernet (10BASE5) requires 250 Vrms. This electrical isolation is normally accomplished by transformer coupling of each signal pair. The kind of transformers recommended for the 82502 are the pulse transformers which have a 1:1 turn ratio and at least 50 microhenry inductance. PE64102 and PE64107 manufactured by Pulse Engineering are found to be good selections for this purpose. The PE 64102 offers 500 Vrms isolation. The PE64107 offers 2000 Vrms isolation. Both products provide three transformers in one package. Even though the current Type 10BASE5 specification requires only 250 Vrms, it is very common to have a higher isolation, at least 500 Vrms, in transceivers.

The standard specifies the voltage input level and maximum current allowed on the power pair of the transceiver cable. The voltage level may be between +11.28V dc and +15.75V dc. The maximum current is limited to 500 mA. Since the 82502 requires +10V ±10% and +5V ±10% as power, there has to be a DC/DC converter. In addition the DC/DC converter must be isolated due to the requirement described above. The DC/DC converter should be able to supply about 100 mA on the +10V line and 60 mA on the 5V line. The efficiency required in the converter is, therefore, ((11V × 100 mA + 5.5V × 60 mA) / ((11.28V − 0.5A × 4Ω) × 500 mA)) × 100 = 31% worst case. 4Ω is the maximum round trip resistance the power pair may have. 82502's CMOS process is the major contributor to this low DC/DC efficiency requirement.

Since the DC/DC converter has an isolation transformer inside, the output voltages are all floating voltages. The 0V output of the converter, for example, has no voltage relationship with the DTE's ground. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the 0V output of the DC/DC converter which is the 82502's ground (reference voltage).

Both Pulse Engineering and Reliability Incorporated produce DC/DC converters that meet the 82502's requirements. The Pulse Engineering's part number is PE64369 (enclosed in this design kit). The device measures about 1.5″ x 1.5″ x 0.5″ and provides 2000 Vrms breakdown. The Reliability's part number is 2E12R10-5. Preliminary data sheets are available from Reliability.

3.10.2 OTHER PASSIVE AND ACTIVE DEVICES FOR THE 82502

A 78Ω ±5% resistor is required to terminate the transmit pair of the Transceiver cable. The chip has an internal circuit that establishes a common mode voltage, thus no voltage divider is required. The receive and collision pair drivers need pull up resistors. A 43.2 ±1% resistor must be connected from each output pin to +5V.

A 243Ω ±0.5% precision resistor is required on the REXT pin to the ground. The accuracy of this resistor is very important since this resistor is a part of current and voltage reference circuits in the analog sections of the 82502.

Grounding the HBD (Heartbeat Disable) pin will allow the chip to perform Signal Quality Error check (Heartbeat) as required by the IEEE 802.3. The chip will transmit the collision presence signal after each transmission during Interframe Spacing (IFS) time. In a repeater application, this feature is disabled (HBD = +5V).

Diodes connected on the CXTD pin are to reduce the capacitive loading onto the coaxial cable. One diode is sufficient, but two will provide a protection in case one burns out (Short Circuit). The diode should have about 2 pF shunt capacitance at Vd = 0V and be able to handle at least 100 mA when biased in forward direction. A few candidates are 1N5282, 1N3600, and 1N4150.

A 100Ω fusible resistor connected on the CXRD pin is purely for protection. It is there as a fuse, not as a resistor. The 82502 works without this resistor. The IEEE 802.3, however, states that "component failures within the MAU (Media Attachment Unit or Transceiver) electronics should not prevent communication among other MAUs on the coaxial cable." It is recommending a transceiver design that minimizes the probability of total network failure. The fusible resistor will provide an open circuit in an event of excess current. A short circuit from the CXRD pin to ground will not bring down the network due to the blown fuse.

A 1 MΩ resistor connected between the coaxial cable shield and the Transceiver cable shield will provide a static discharge path. The Ethernet coaxial cable should also have an effective earth ground at one point in a network as required by the standard. A 0.01 μF in parallel to the 1 MΩ resistor provides ground for RF signals.

3.10.3 LAYOUT CONSIDERATION FOR THE 82502 CIRCUITS

It is strongly recommended that the board have a special ground plane for the 82502 (see Figure 11). The 0V (reference) output of the isolated DC/DC converter should be connected to the ground plane. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the ground plane with minimum lead wires.

There should be a 0.22 μF capacitor connected between the coaxial cable shield and ground. The signal path from the coax. shield through the 0.22 μF capacitor to

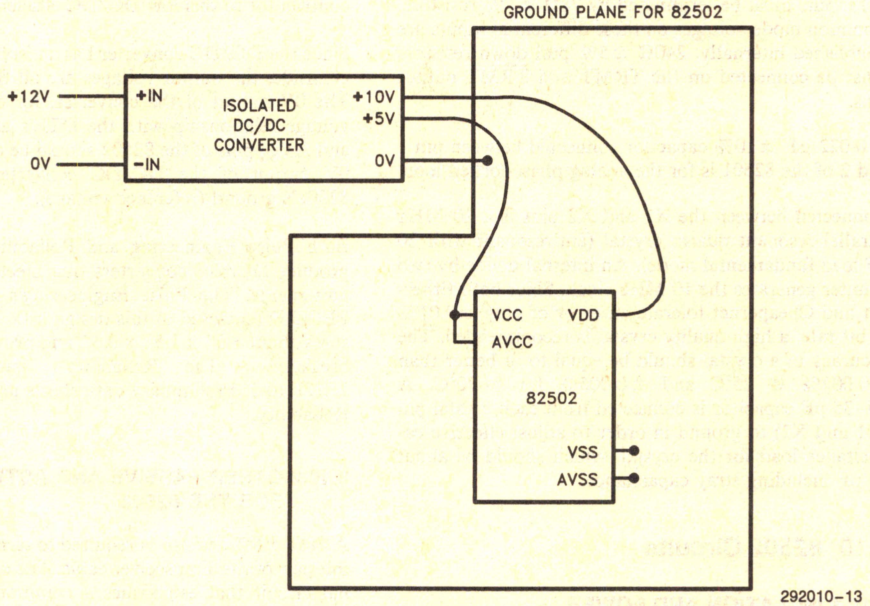


Figure 11. Ground Plane for the 82502

the ground should be kept as short as possible—leads of the 0.22 μF capacitor should be as short as possible.

The path length from the CXTD pin through two diodes to the center conductor of the coax should also be minimized.

These are recommendations which will produce a more reliable circuit if followed carefully. Remember that the 82502 has analog circuits in it.

4.0 DEMONSTRATION SOFTWARE

The demonstration software included in this Ap Note is called "Traffic Simulator and Monitor Station" (TSMS) program. The TSMS program is written in PL/M and has the following features:

1. Programmable network load generation
2. Network statistical monitoring capabilities
3. Interactive command execution of all 82586 commands
4. Interactive buffer monitoring

The environment created with the TSMS program was found to be very useful for network debugging and other individual station's hardware and software debugging. The TSMS software listing is found in Appendix B.

NOTE:
The 82586 Date Link Driver presented in AP Note 235 also runs on the LANHIB. Please refer to the Ap Note for detailed operations of the software.

4.1 Programming PROMs to Run the TSMS Program

By returning the card enclosed in this kit or by contacting Insite, the TSMS program and related batch files can be obtained on a diskette. TSMS related files that are on the diskette are:

READ.ME
TSMS.PLM
IO.PLM
INI186.PLM
LANHIB.BAT
SBC.BAT
IUPHIB.BAT
IUPSBC.BAT
HI.BYT
LO.BYT
ROM.BAT

The READ.ME file contains instructions for programming PROMs. HI.BYT and LO.BYT are the files which can be downloaded to PROMs directly. These files are already configured for the LANHIB. The batch file ROM.BAT invokes the Intel PROM Programming Software (iPPS) under the DOS operation system and programs two 2764 EPROMs. The Intel Universal Programmer must be placed in ON-LINE mode.

Other files contained in the diskette are for compiling and locating the original TSMS program. Using these files, the original TSMS program can be changed or can be compiled for an iSBC 186/51. 'TSMS.PLM' is the original TSMS source program. 'IO.PLM' contains the IO driver needed when the TSMS program is run on the iSBC 186/51. INI186.PLM is the LANHIB initialization routine. LANHIB.BAT is the batch file that compiles, links, and locates the TSMS program and the LANHIB initialization routine. SBC.BAT compiles, links, and locates the TSMS program and the IO driver for the iSBC 186/51. IUPHIB.BAT programs two 2764s for the LANHIB. IUPSBC.BAT programs two 2764s for the iSBC 186/51.

Therefore, if the TSMS program is to be run on the LANHIB (Demo board), steps required are:

1. C:>LANHIB
2. C:>IUPHIB

If the TSMS program is to be run on the iSBC 186/51, steps required are:

1. C:>SBC
2. C:>IUPSBC

4.2 Capabilities and Limits of the TSMS Program

The TSMS program initializes the LANHIB Ethernet/Cheapernet station by executing 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands. The program asks a series of questions in order to set up a linked list of these 82586 commands. After initialization is completed, the program automatically starts the 82586's Receive Unit (monitoring capability). Transmissions are optional (traffic simulation capability).

The TSMS program has two modes of operation: Continuous mode and Interactive Command Execution mode. The program automatically gets into the Continuous mode after initialization. The Interactive Command Execution mode can be entered from the Continuous mode. Once entered in the Continuous mode, the software uses the format shown in Figure 12 to display information. Detailed description of each of these fields is as follows:

Host Address: host (station) address used in the most recently prepared IA-Setup command. The software simply writes the address stored in the IA-Setup command block with its least significant bit being in the most right position. Note that if the IA-Setup com-

```
****************************** Station Configuration *************************

Host Address: 00  AA  00  00  18  6D
Multicast Address(es): No Multicast Addresses Defined
Destination Address: FF  FF  FF  FF  FF  FF
Frame Length: 118 bytes
Time Interval between Transmit Frames:  159.4 microseconds
Network Percent Load generated by this station: 35.7 %
Transmit Frame Terminal Count: Not Defined
82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

****************************** Station Activities ****************************
```

# of Good Frames Transmitted	# of Good Frames Received	CRC Errors	Alignment Errors	No Resource Errors	Receive Overrun Errors
10130	0	0	0	0	0

292010-14

Figure 12. Continuous Mode Display

mand was just set up and not executed, the address displayed in this field may not be the address stored in the 82586.

Multicast Address(es): multicast addresses used in the most recently prepared MC-Setup command. As in the case of host address, the software simply writes the addresses stored in the MC-Setup command block. Note that if the MC-Setup command was just set up and not executed, the addresses displayed in this field may not be the addresses stored in the 82586.

Destination Address: destination address stored in the transmit command block if AL-LOC = 0. If AL-LOC = 1, destination address is picked up from the transmit buffer. The least significant bit is in the most right position.

Frame Length: transmit frame byte count including destination address, source address, length, data, and CRC field.

Time Interval Between Transmit Frames: approximate time interval obtainable between transmit frames (Figure 13). The number is correct if there are no other stations transmitting on the network.

Network Percent Load Generated by This Station: approximate network percent load that is generated by this station (Figure 13). The number is correct if there are no other stations transmitting on the network.

Transmit Frame Terminal Count: number of frames this station will transmit before it stops network traffic load generation. If this station is transmitting indefinitely, this field will be 'Not Defined'.

82586 Configuration Block: configuration parameters used in the most recently prepared Configure command. As in the case of IA-Setup command, the software simply writes the parameters from the Configure command block. The least significant byte (FIFO Limit) of the configuration parameters is printed in the most left position.

# of Good Frames Transmitted: number of good frames transmitted. This is a snap shot of the 32-bit transmit frame counter. It is incremented only when both C and OK bits of the transmit command status are set after an execution. The counter is 32-bit wide.

# of Good Frames Received: number of good frames received. This is a snap shot of the 32-bit receive frame counter. It is incremented only when both C and OK bits of a receive frame descriptor status are set after a reception. The counter is 32-bit wide.

CRC Errors: number of frames that had a CRC error. This is a snap shot of the 16-bit CRC counter maintained by the 82586 in the SCB.

Alignment Errors: number of frames that had an alignment error. This is a snap shot of the 16-bit alignment counter maintained by the 82586 in the SCB.

No Resource Errors: number of frames that had a no resource error. This is a snap shot of the 16-bit no resource counter maintained by the 82586 in the SCB.

Receive Overrun Errors: number of frames that had a receive overrun error. This is a snap shot of the 16-bit receive overrun error counter maintained by the 82586 in the SCB.

If the station is actively transmitting, # of good frames transmitted should be incrementing. If the station is actively receiving frames, # of good frames received should be incrementing. In this continuous mode, a user can see the activities of the network.

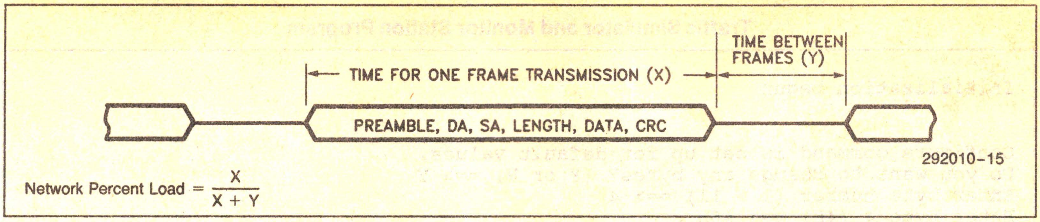


Figure 13. Network Percent Load

Hitting any key on the keyboard while the program is running in the Continuous mode will exit the mode. The program will respond with a message 'Enter Command (H for Help) —> '. In this Interactive Command Execution mode, a user can set up any one of the 82586 action commands and/or execute any one of the 82586 SCB control commands. Setting up a Dump command and executing a SCB Command Unit Start command will, for example, execute the Dump command. Display commands are also available to see the contents of the 82586's data structure blocks. A display command will enable a user to see the contents of the 82586's dump (see Section 6.3).

Typing 'E' after 'Enter command (H for help) —> ', executing a SCB Command Unit Start command with a transmit command, or executing a SCB Receive Unit Start command will exit the Interactive Command Execution mode. The program will be back in the Continuous mode. Using this Interactive Command Execution mode, one can, for example, reconfigure the station and come back to the Continuous mode. Section 6 lists actual example executions of the TSMS program.

The TSMS program should be run in an 8 MHz system. The software running at 8 MHz with a maximum of 2 wait states has been tested and verified to be able to receive back-to-back frames separated by 9.6 microseconds and still keep track of the correct number of frames received. This capability, for example, can be used to find out exactly how many frames a new station in the network had transmitted.

The software does not perform extensive loopback tests and hardware diagnostics during the initialization. A loopback operation can be performed interactively in the Interactive Command Execution mode.

The software allows a user to set up only 8 multicast addresses maximum. It is not possible with this program to set up more than 8 multicast addresses.

The command chaining feature of the 82586 is not used in the Interactive Command Execution mode. Each command setup performed by a 'S' command after 'Enter command (H for help) —> ' sets up a command with its EL bit set, I bit reset, and S bit reset. Diagnose, Configure, IA-Setup, and MC-Setup commands are chained together during the initialization routine and executed at once with only one CA.

The software sets up 5 Receive Frame Descriptors linked in a circular list. Therefore, a user can see only the last 5 frames the station has received. It also sets up 5 receive buffers, each being 1514 bytes long, linked in circle. Therefore, the 82586 never goes into the NO RESOURCES state.

4.3 Example Executions of the TSMS Program

This section presents three example executions of the TSMS program. When the TSMS program needs a command to be typed, it asks a question with ' —> '. Anything after ' —> ' is what a user needs to type in on the keyboard. To switch from the continuous mode to the interactive command execution mode, type any key on the keyboard.

4.3.1 EXAMPLE 1: EXTERNAL LOOPBACK EXECUTION

In this example, 500 external loopback transmissions and receptions are executed (Figure 14). In order for the software to process each loopback properly, a large delay was given between transmissions.

4.3.2 EXAMPLE 2: FRAME RECEPTION IN PROMISCUOUS MODE

The 82586 is configured to receive any frame that exists in the network (Figure 15). In this example, the station received 100 frames.

4.3.3 EXAMPLE 3: 35.7% NETWORK TRAFFIC LOAD GENERATION

The station is programmed to transmit 118 byte long frames with a time interval of 159.4 microseconds in between (Figure 16). The network load is about 35.7 percent if no other stations are transmitting in the network.

A key was hit to enter the Interactive Command Execution mode. In that mode, a Dump command was executed and the result was displayed. After the Dump execution, a transmit command was set up again and the station was put in the Continuous mode.

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 10000000000
The number is too big.
It has to be less than or equal to 65535 (FFFFH).
Enter a number ==> 60000

Setup a transmit terminal count? (Y or N) ==> Y
Enter a transmit terminal count ==> 500

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 30.18 miliseconds
Network Percent Load generated by this station:` .0 %
Transmit Frame Terminal Count: 500

Good enough? (Y or N) ==> Y

Receive Unit is active.
```

292010-16

Figure 14. External Loopback Execution

```
---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
2200
0000
0000
0000


Hit <CR> to countinue

transmission started!




****************************** Station Configuration *************************

 Host Address: 00  00  00  00  22  00
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: 00  00  00  00  22  00
 Frame Length: 20 bytes
 Time Interval between Transmit Frames: 30.18 miliseconds
 Network Percent Load generated by this station:   .0 %
 Transmit Frame Terminal Count: 500
 82586 Configuration Block: 08  00  A6  00  60  00  F2  00  00  06

******************************* Station Activities ****************************

 # of Good     # of Good     CRC          Alignment     No            Receive
 Frames        Frames        Errors       Errors        Resource      Overrun
 Transmitted   Received                                 Errors        Errors
 500           500           0            0             0             0
```

292010–17

Figure 14. External Loopback Execution (Continued)

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 9
Enter byte 9 (9H) ==> 1
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> Y
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> N

Receive Unit is active.






****************************** Station Configuration ************************

Host Address: 00  AA  00  00  18  6D
Multicast Address(es): No Multicast Addresses Defined
82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

******************************* Station Activities **************************

# of Good     # of Good     CRC           Alignment     No            Receive
Frames        Frames        Errors        Errors        Resource      Overrun
Transmitted   Received                                  Errors        Errors
0             100           0             0             0             0
Enter command (H for help) ==> D

Command Block or Receive Area? (R or C) ==> R
Frame Descriptors:
4000 at 036C  A000 at 0382  A000 at 0398  A000 at 03AE  A000 at 03C4
0000          0000          0000          0000          0000
0382          0398          03AE          03C4          036C
03DA          03E4          03EE          03F8          0402
2200          2200          2200          2200          2200
2200          2200          2200          2200          2200
0000          0000          0000          0000          0000
```

292010-18

Figure 15. Frame Reception in Promiscuous Mode

```
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000

 Receive Buffer Descriptors:
C064 at 03DA  C064 at 03E4  C064 at 03EE  C064 at 03F8  C064 at 0402
03E4          03EE          03F8          0402          03DA
040C          09F6          0FE0          15CA          1BB4
0000          0000          0000          0000          0000
05DC          05DC          05DC          05DC          05DC

 Display the receive buffers? (Y or N) ==> Y
 Receive Buffers:

 Receive Buffer 0 :
002C:014C 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:015C 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:016C 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:017C 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:018C 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:019C 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:01AC 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 1 :
002C:0736 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0746 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0756 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0766 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0776 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0786 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0796 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 2 :
002C:0D20 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0D30 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0D40 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0D50 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0D60 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0D70 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0D80 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 3 :
002C:130A 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:131A 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:132A 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:133A 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:134A 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:135A 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:136A 60  61  62  63

Hit <CR> to countinue
```

292010–19

Figure 15. Frame Reception in Promiscuous Mode (Continued)

```
 Receive Buffer 4 :
002C:18F4 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:1904 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:1914 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:1924 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:1934 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:1944 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:1954 60  61  62  63

Hit <CR> to countinue


 Enter command (H for help) ==> E





**************************** Station Cofiguration ****************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

***************************** Station Activities ****************************

 # of Good     # of Good    CRC        Alignment    No          Receive
 Frames        Frames       Errors     Errors       Resource    Overrun
 Transmitted   Received                             Errors      Errors
 0             100          0          0            0           0
```

292010-20

Figure 15. Frame Reception in Promiscuous Mode (Continued)

Traffic Simulator and Monitor Station Program

```
Initialization begun

Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> Y
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> FFFFFFFFFFFF

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 100
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N

Enter a delay count ==> 0

Setup a transmit terminal count? (Y or N) ==> N

Destination Address: FF  FF  FF  FF  FF  FF
Frame Length: 118 bytes
Time Interval between Transmit Frames:  159.4 microseconds
Network Percent Load generated by this station: 35.7 %
Transmit Frame Terminal Count: Not Defined

Good enough? (Y or N) ==> Y

Receive Unit is active.

---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
FFFF
FFFF
FFFF
0000

Hit <CR> to countinue
```

292010-21

Figure 16. 35.7% Network Load Generation

```
transmission started!


****************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00 40

****************************** Station Activities ***************************

 # of Good     # of Good     CRC           Alignment     No            Receive
 Frames        Frames        Errors        Errors        Resource      Overrun
 Transmitted   Received                                  Errors        Errors
 10459         0             0             0             0             0
 Enter command (H for help) ==> H

 Commands are:
 S - Setup CB                D - Display RFD/CB
 P - Print SCB               C - SCB Control CMD
 L - ESI Loopback On         N - ESI Loopback Off
 A - Toggle Number Base
 Z - Clear Tx Frame Counter
 Y - Clear Rx Frame Counter
 E - Exit to Continuous Mode


 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> H

Command block type:
 N - Nop                   I - IA Setup
 C - Configure             M - MA Setup
 T - Transmit              R - TDR
 D - Diagnose              S - Dump Status
 H - Print this message

 Enter command block type (H for help) ==> S

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention



 Enter command (H for help) ==> D
```

292010-22

Figure 16. 35.7% Network Load Generation (Continued)

```
 Command Block or Receive Area? (R or C) ==> C
---Dump Status Command Block---
A000 at 0364
8006
FFFF
27D6

 Dump Status Results
 at 27D6
00  E8  3F  26  08  60  00  FA  00  00  40  FF  6D  18  00  00
AA  00  40  20  00  00  00  00  FF  FF  FF  FF  B5  9E  EE  CF
62  63  3F  B0  00  00  00  00  00  00  00  00  FF  85  08  FC
00  00  00  00  00  00  00  00  00  00  00  00  70  03  06  00
DC  05  00  00  0C  04  DC  05  E4  03  DA  03  DA  03  78  05
82  03  6C  03  F8  03  64  80  D6  27  E8  21  FF  FF  4E  03
06  80  FF  FF  64  03  00  00  D2  02  00  00  00  00  00  00
00  00  D6  27  00  01  00  28  00  00  00  00  30  26  00  00
20  00  40  06  30  01  00  00  90  00  10  01  00  00  6C  03
00  00  6A  03  0E  00  6C  28  00  00  74  03  00  00  00  00
00  00  00  00  00  C0  00  00  00  00

 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> T
 Enter a destination address in Hex ==> FFFFFFFFFFFF

 Enter TYPE ==> 0
 How many bytes of transmit data?
 Enter a number ==> 100
 Transmit Data is continuous numbers (0, 1, 2, 3, ... )
 Change any data bytes? (Y or N) ==> N

 Enter a delay count ==> 0

 Setup a transmit terminal count? (Y or N) ==> N

 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined

 Good enough? (Y or N) ==> Y

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention
```

292010-23

Figure 16. 35.7% Network Load Generation (Continued)

```
****************************** Station Configuration *************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

****************************** Station Activities ***************************

 # of Good     # of Good     CRC           Alignment     No            Receive
 Frames        Frames        Errors        Errors        Resource      Overrun
 Transmitted   Received                                  Errors        Errors
 106020        0             0             0             0             0
                                                                   292010-24
```

Figure 16. 35.7% Network Load Generation (Continued)

5.0 IN CASE OF DIFFICULTY

This section presents methods of troubleshooting ("debugging") a LANHIB board. When a LANHIB board is powered up with the TSMS program stored in EPROMs, it should display "TRAFFIC SIMULATOR AND MONITOR STATION PROGRAM" message on a terminal screen. If the message is not displayed, the board has to be debugged. Section 5.1 describes basic 80186/82586 system troubleshooting procedures. Section 5.2 is for troubleshooting 82501 and 82502 circuits. After the 80186/82586 system is debugged, the 82501/82502 circuits have to be tested.

5.1 Troubleshooting 80186/82586 System

Shown in Figure 17 is a flow chart for troubleshooting 80186/82586 system. The procedure requires an oscilloscope. A logic analyzer is needed if problems appear to be serious. The procedures will debug the board to the point where the 82530 is initialized properly. If the 82530 can be initialized properly, ROM and RAM interfaces must be functioning. Board initialization routines (INI186.PLM) linked to the TSMS program requires ROM and RAM accesses. Since the 82586 shares most of the system with the 80186, no special debugging is required for the 82586. Wiring of all 82586 parallel signal pins should, however, be checked.

The flow chart branches to two major paths after the first decision box. One path debugs the RS-232 channel and the other debugs the 80186/82586 system. The waveform of the $\overline{\text{TRXCB}}$ output of the 82530 determines which path to be taken. If the 82530 is getting programmed properly, there should be 153.6 KHz (1/f = 6.51 μs) clock on this output pin. If there is a clock, the problem is probably in the RS-232 interface. If there is no clock, then the system has to be debugged using a logic analyzer.

5.2 Troubleshooting 82501/82502 Circuits

If the TSMS program runs on the LANHIB but the 82586 is not able to transmit or receive, there must be a problem in 82501/82502 circuits. The flow chart in Figure 19 will guide troubleshooting in these circuits. An oscilloscope is required.

The board should be configured to Cheapernet and disconnected from the network. Two terminators will be required to terminate a "T" BNC connector providing an effective load resistance of 25Ω to the 82502.

The 82586 must have the system and transmit clocks running upon reset. Since the transmit clock is generated by the 82501, the 82501 transmit clock output pin (pin 16) should be checked. The TSMS program executes 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands during initialization. If the 82586 has active $\overline{\text{CRS}}$ (Carrier Sense) signal, it cannot complete execution of these commands. The 82501 should, therefore, be checked if it is generating inactive $\overline{\text{CRS}}$ signal to the 82586 after power up. The LANHIB powers up the 82501 in non-loopback mode.

After making sure that the 82501 is generating proper signals to the 82586, the TSMS program is restarted with an initialization shown in Figure 20. The 82586 is configured to EXT-LPBK = 1, TONO-CRS = 1, and MIN-FRM-LEN = 6. The chip is also loaded with a destination address identical to the source address. If there are no problems in the 82501/82502 circuits, the station will be receiving its own transmitted frames. If problems exist, the station will only be transmitting. Since the 82586 is configured to TONO-CRS (Transmission On NO Carrier Sense), the chip will keep transmitting regardless of the state of carrier sense. The 82501/82502 circuits can then be probed with an oscilloscope at the locations indicated in Figure 21. Probing will catch problems like wiring mistakes, missing load resistors, etc.

Once the station is debugged, it can be connected to the network. If there is a problem in the network, the 82586's TDR command can be used to find the location and nature of the problem.

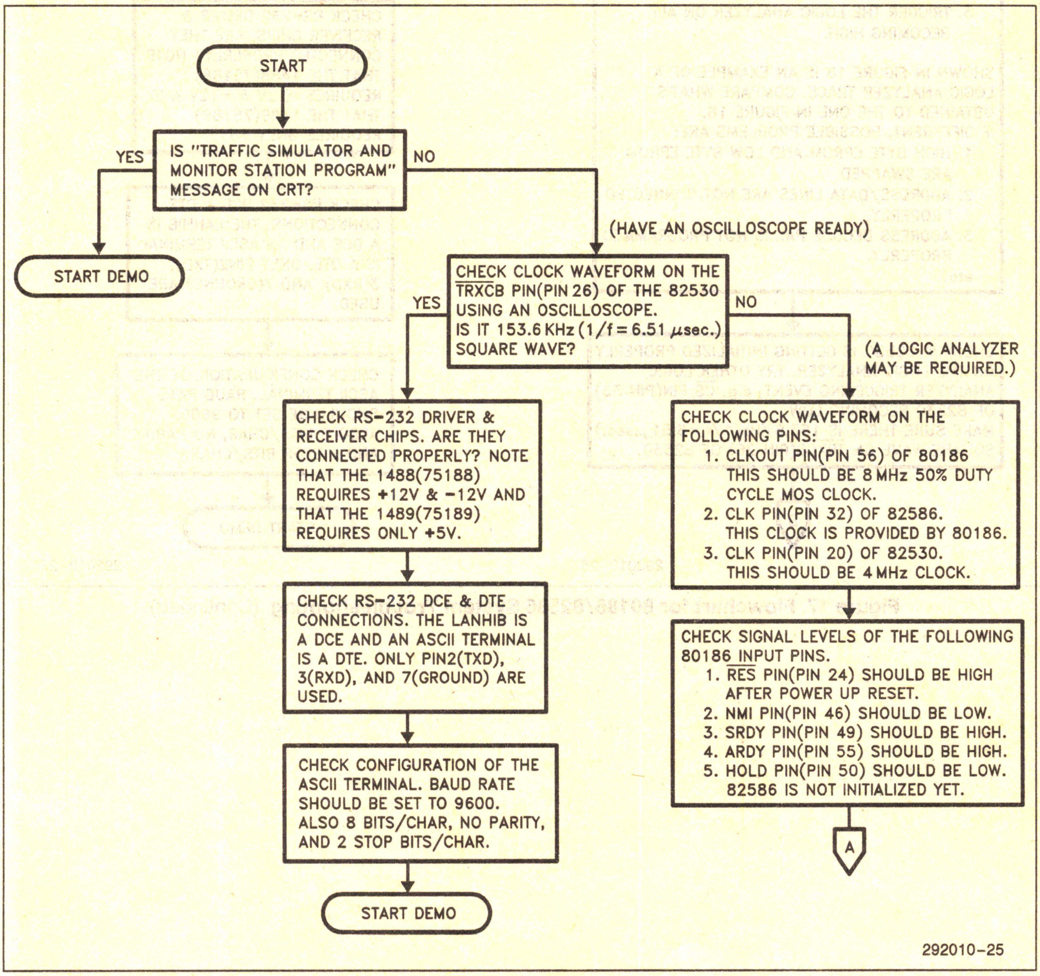


Figure 17. Flowchart for 80186/82586 System Troubleshooting

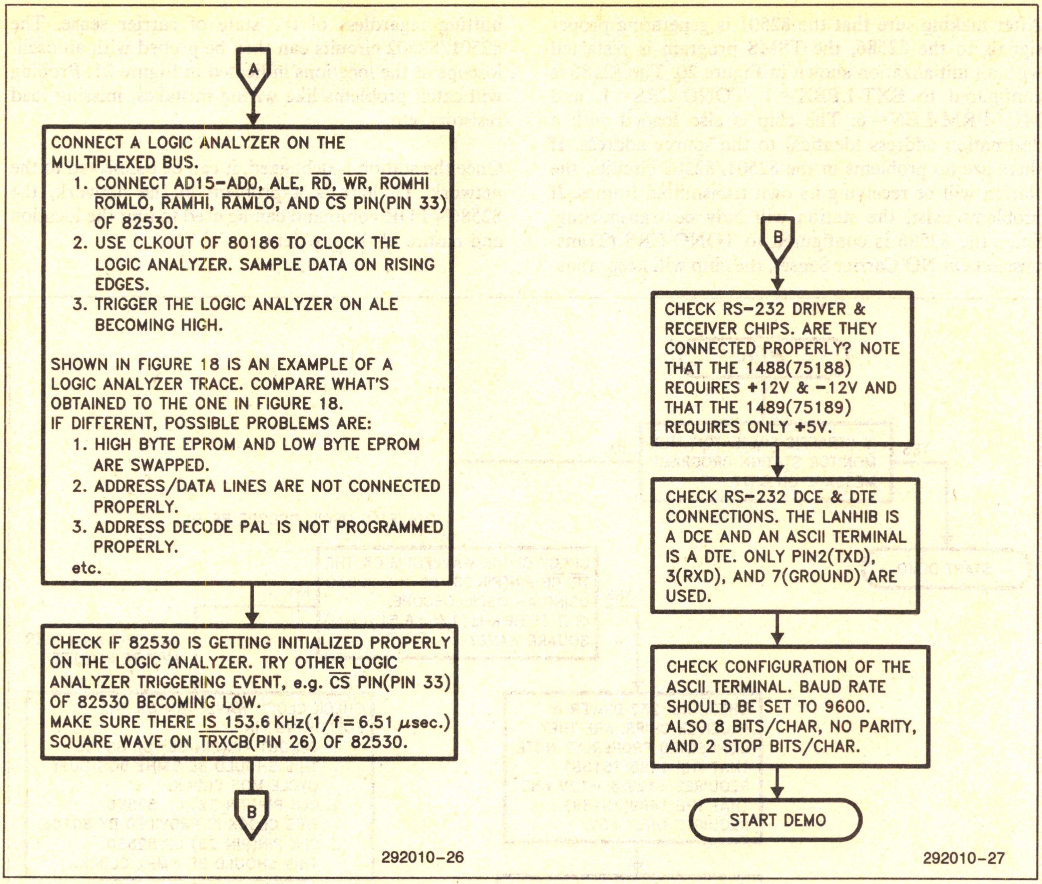


Figure 17. Flowchart for 80186/82586 System Troubleshooting (Continued)

```
                                  AD15-AD0
                                  ALE
                                  RD#
                                  WR#
                                  ROMHI#
                                  ROMLO#      THESE ARE MEMORY CHIP SELECTS
                                  RAMHI#
                                  RAMLO#
                                  CS# PIN (PIN 33) OF 82530

0097  00 41  01001111
0098  00 41  01001111
0099  00 41  01101111
TRIG  00 41  11101111 ..... LOGIC ANALYZER IS TRIGGERED ON ALE = HI.
0101  FF F0  01001111 ..... 80186 JUMPS TO FFF0H AFTER RESET.
0102  06 EA  00101111 ..... JMP INSTRUCTION (DIRECT INTERSEGMENT)
0103  06 EA  00101111       SEGMENT OFFSET = 0006H
0104  06 EA  00101111       SEGMENT SELECTOR = FFC0H
0105  06 EA  00101111       (80186 INSERTS 3 WAIT STATES BEFORE
0106  06 EA  00101111        UMCS REGISTER IS PROGRAMMED.)
0107  06 EA  11101111
0108  FF F2  01101111
0109  C0 40  00101111
0110  C0 00  00101111
0111  C0 00  00101111
0112  C0 00  00101111
0113  C0 00  00101111
0114  C0 00  11101111
0115  FF F4  01101111
0116  FF FF  00101111
0117  FF FF  00101111
0118  FF FF  00101111
0119  FF FF  00101111
0120  FF FF  00101111
0121  FF FF  11101111
0122  FF F6  01101111
0123  00 40  00101111
0124  00 00  00101111
0125  00 00  00101111
0126  00 00  00101111
0127  00 00  00101111
0128  00 00  11101111
0129  FC 06  01101111 ..... JUMPED TO FC06H
0130  2E FA  00101111
0131  2E FA  00101111
0132  2E FA  00101111
0133  2E FA  00101111
0134  2E FA  00101111
0135  2E FA  11101111
0136  FC 08  01101111
0137  16 8E  00101111
0138  16 8E  00101111
```

292010-28

Figure 18. Example of Logic Analyzer Trace

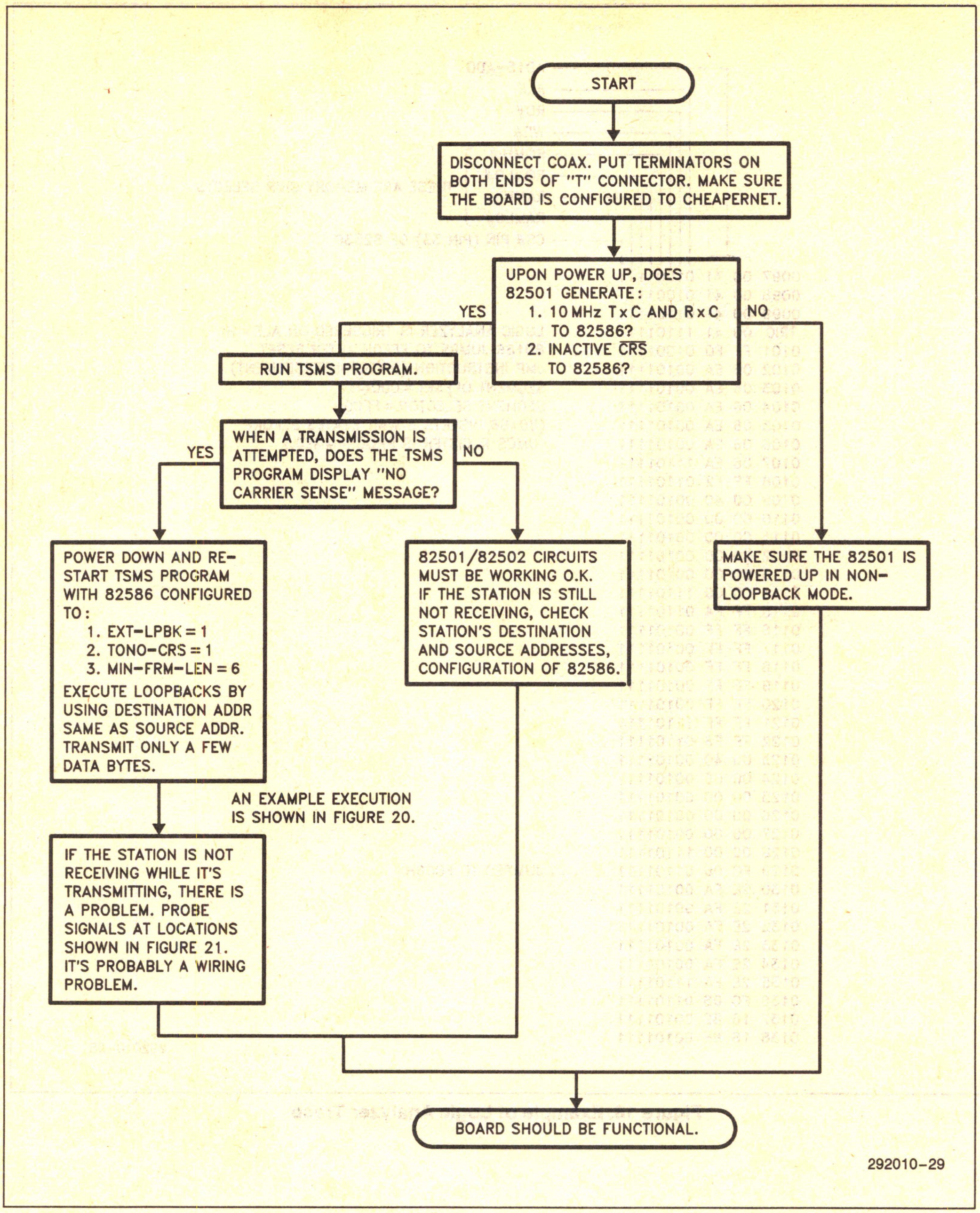


Figure 19. Flowchart for 82501/82502 Circuits Troubleshooting

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 9
Enter byte 9 (9H) ==> 08H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 0
Setup a transmit terminal count? (Y or N) ==> N

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 159.4 seconds
Network Percent Load generated by this station: 11.0 %
Transmit Frame Terminal Count: Not Defined

Good enough? (Y or N) ==> Y
```

292010-77

Figure 20. TSMS Initialization for 82501/82502 Circuits Troubleshooting

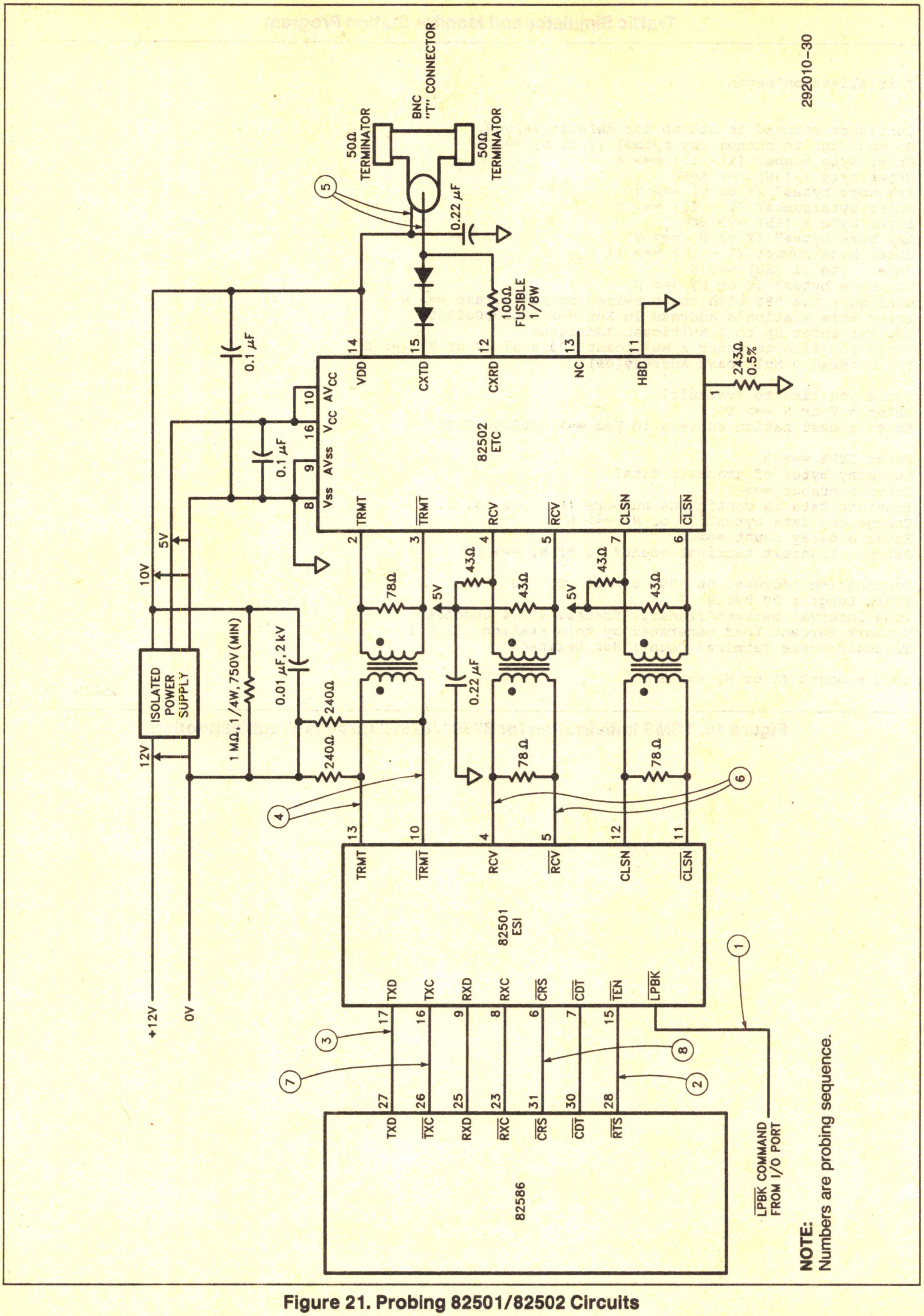
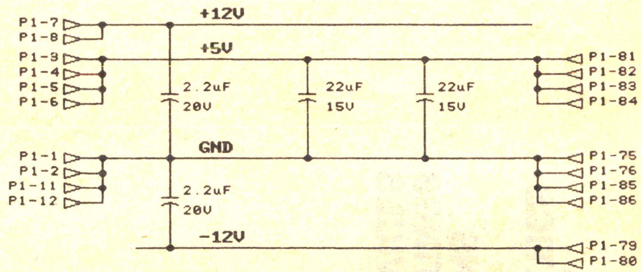


Figure 21. Probing 82501/82502 Circuits

APPENDIX A
LANHIB SCHEMATICS
PARTS LIST
PAL EQUATIONS
DIP SWITCH SETTINGS
WIRE WRAP SERVICES

PARTS LIST

REFERENCES	DESCRIPTION	MFR. PART NO.	MFR. CODE	QTY.
U1	IC	74S32	OBD	1
U2	IC	74LS04	OBD	1
U3, U4	IC	74LS245	OBD	2
U5	IC	74F373	OBD	1
U6, U7	IC	74LS373	OBD	2
U8	IC	80186	INT	1
U9	IC	82586	INT	1
U10	IC	16L8	OBD	1
U11	IC	74LS02	OBD	1
U12, U27	IC	74LS74	OBD	2
U28	IC	74AS74	OBD	1
U13	IC	74LS165	OBD	1
U14	IC	82501	INT	1
U15	Pulse Transformer Pack	PE64102	PE	1
U16	IC	82502	INT	1
U17	DC/DC Converter	PE64369	PE	1
U18, U20	IC, 64K-Bit EPROM	2764-20	INT	2
U26	IC	74AS08	OBD	1
U22, U23	IC, SRAM	HM6264-15	HIT	2
U24	IC, 256-Bit PROM	TBP18S030	TI	1
U25	IC	74AS04	OBD	1
U29	IC	82530	INT	1
U30	IC	1489	OBD	1
U31	IC	1488	OBD	1
U32	IC, 1M-Bit EPROM (Optional)	27210	INT	1
R1-R3, R6 R19, R20	Resistor, 10K ohm, 1/4W, 5%	COML	OBD	6
R4	Resistor, 100K ohm, 1/4W, 5%	COML	OBD	1
R5	Resistor, 2.2K ohm, 1/4W, 5%	COML	OBD	1
R7, R8, R12	Resistor, 78.7 ohm, 1/8W, 1%	COML	OBD	3
R9, R10	Resistor, 240 ohm, 1/4W, 5%	COML	OBD	2
R11	Resistor, 1M ohm, 1/4W, 750Vdc (min), 5%	COML	OBD	1
R13-R16	Resistor, 43.2ohm 1/8W, 1%	COML	OBD	4
R17	Resistor, 100 ohm, Fusible 1/8W, 5%	COML	RCD	1
R18	Resistor, 243 ohm, 1/8W, 0.5%	COML	OBD	1
R21, R22	Resistor, 5K ohm, 1/4W, 5%	COML	OBD	2
RP1	Resistor Pack, 1K ohm, 16 pin	COML	OBD	1
R23-R26	Resistor, 1K ohm, 1/4W, 5%	COML	OBD	4
C1, C2	Capacitor, 20pF, 100V, 5%	COML	OBD	2
C3	Capacitor, 10uF, 20V	COML	OBD	1
C4, C5	Capacitor, 30pF, 100V, 5%	COML	OBD	2
C6	Capacitor, 0.022uF, 50V	COML	OBD	1
C7	Capacitor, 1.0uF, 50V	COML	OBD	1
C11, C12	Capacitor, 0.01uF, 50V	COML	OBD	2
C10	Capacitor, 0.01uF, 2KV	COML	OBD	1
C8, C9	Capacitor, 0.22uF, 50V	COML	OBD	2
CR1	Diode	1N914	OBD	1
CR2, CR3	Diode	1N5282	OBD	2
Y1	Parallel Resonant Crystal, 16M Hz	COML	OBD	1
Y2	Parallel Resonant Crystal, 20M Hz	COML	OBD	1



POWER SUPPLY CONNECTIONS

NOTES:

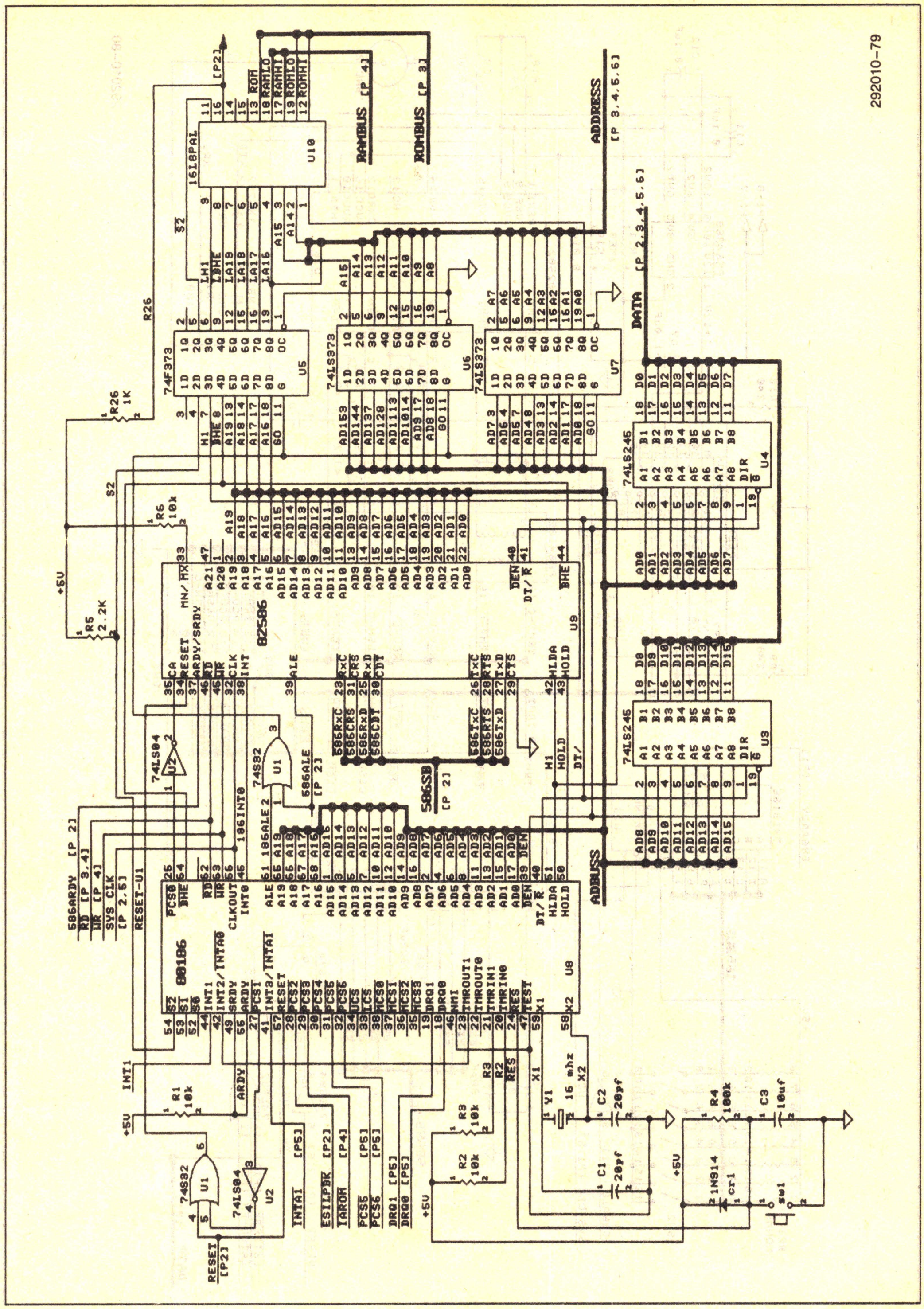
1. THE BOARD REQUIRES +5V, +12V, AND -12V. MULTIBUS POWER PINS FOR THESE VOLTAGES AND GROUND ARE SHOWN ABOVE.

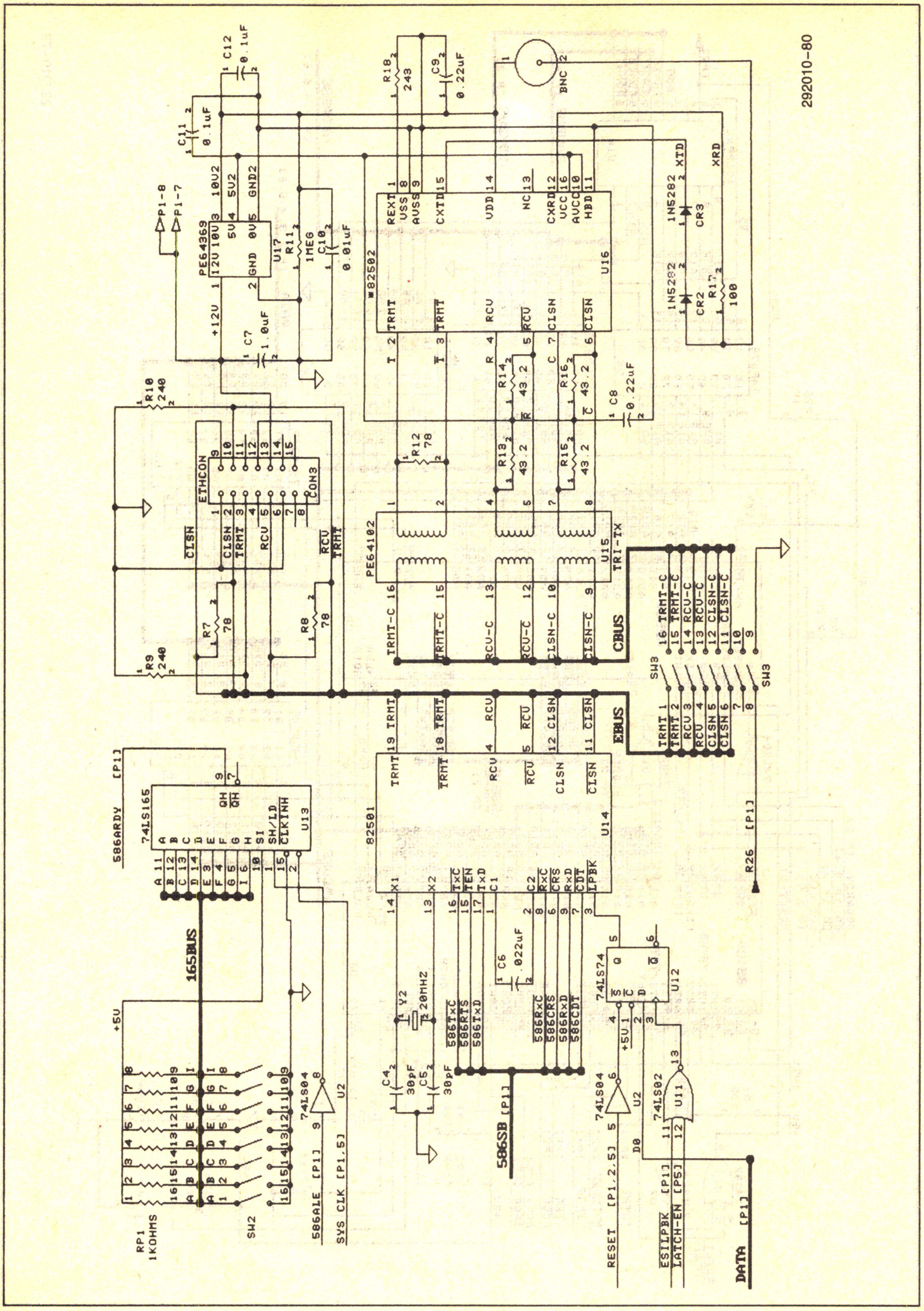
2. EACH IC SHOULD HAVE A 0.1uF CAPACITOR BETWEEN POWER PIN AND GROUND PIN. PARTS LIST DOES NOT INCLUDE DECOUPLING CAPACITORS.

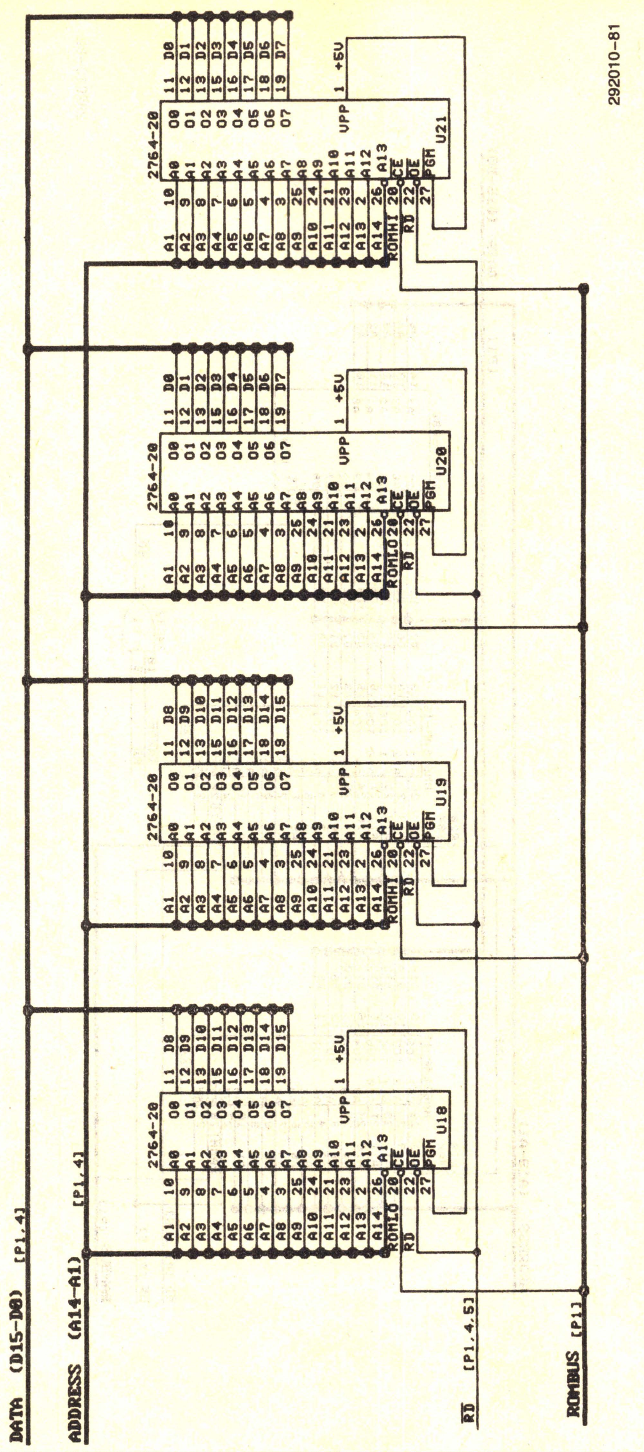
3.

MTR. CODE	MANUFACTURE	LOCATION
INT	INTEL CORPORATION	SANTA CLARA, CA
HIT	HITACH AMERICA LTD.	SAN JOSE, CA
OBD	ORDER BY DESCRIPTION (ANY COMMERCIAL (COML) SOURCE)	
PE	PULSE ENGINEERING	SAN DIEGO, CA
RCD	RCD COMPONENTS INC.	MANCHESTER, NH
TI	TEXAS INSTRUMENTS	DALLAS, TX

292010-78

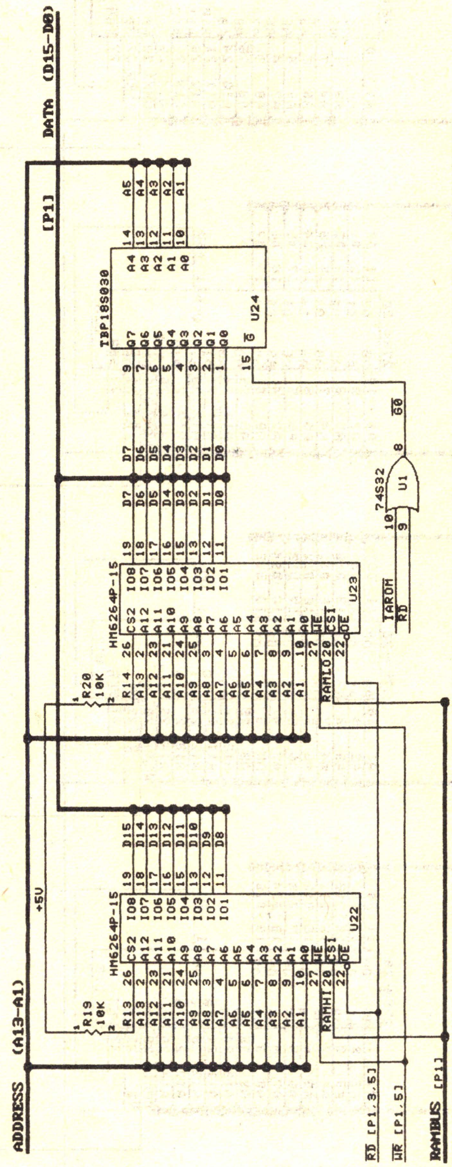

80186
U8
82586
U9
74F373
U5
74LS373
U6
74LS373
U7
16L8PAL
U10
74LS245
U4
74LS245
U3
RAMBUS [P 4]
ROMBUS [P 3]
ADDRESS [P 3,4,5,6]
DATA [P 2,3,4,5,6]
ADDBUS
586SB [P 2]
16 mhz
292010-79

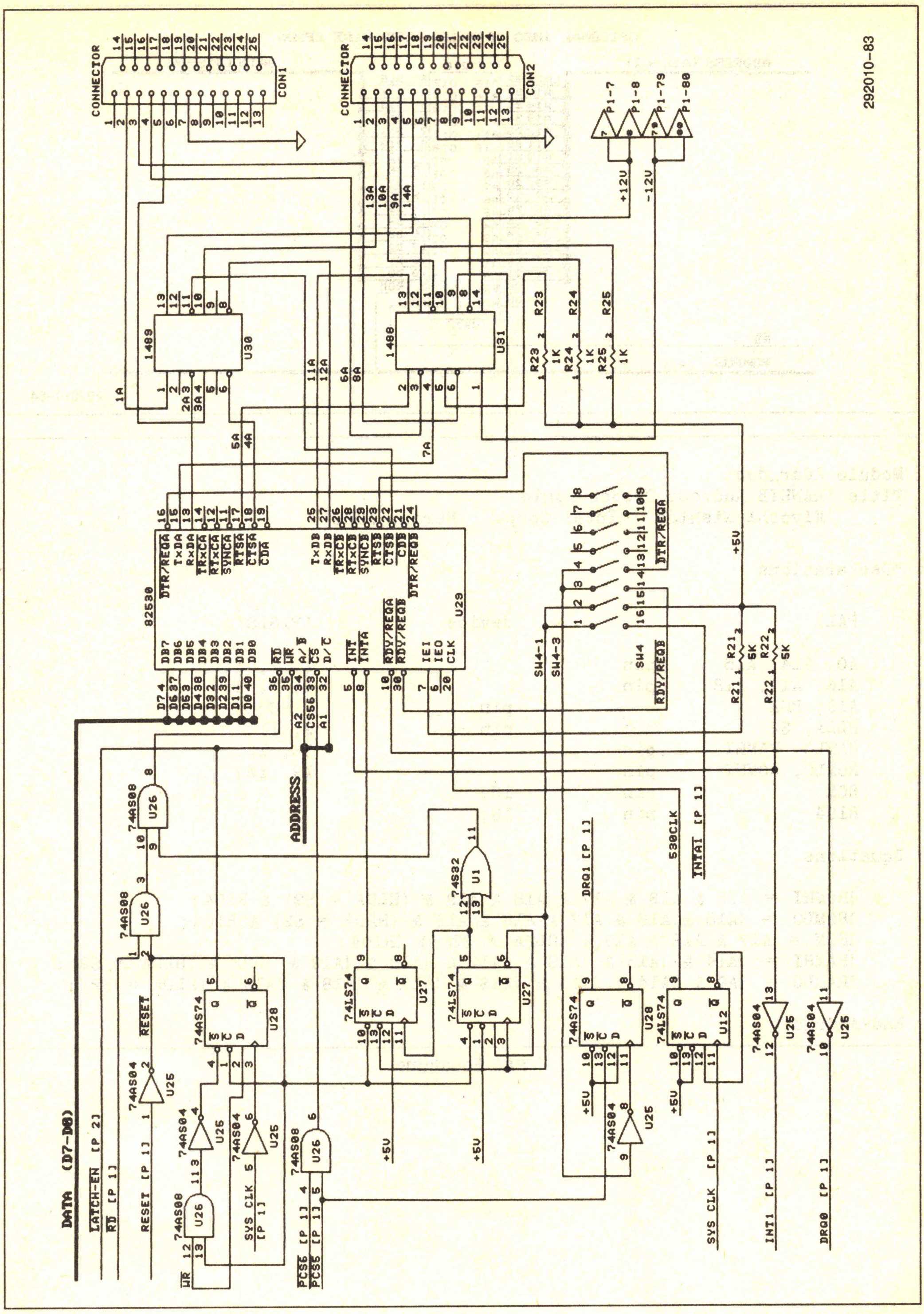

292010-80

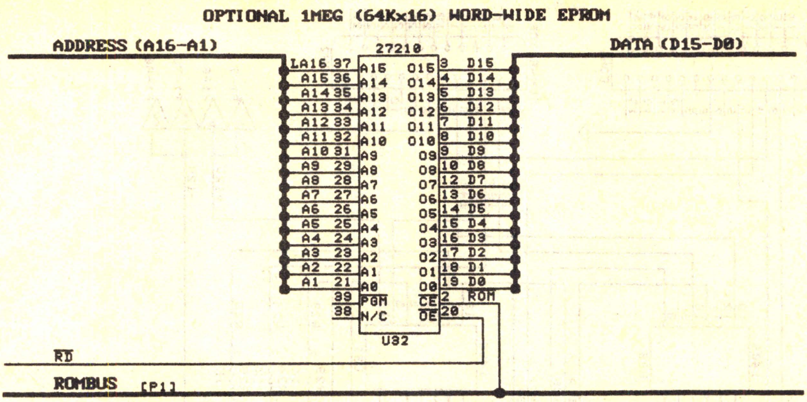

DATA (D15-D0) [P1,4]
ADDRESS (A14-A1) [P1,4]
RD [P1,4,5]
ROMBUS [P1]
2764-20
U18
U19
U20
U21
ROMLO
ROMHI
+5V
VPP
CE
OE
PGM

292010-81

292010-82


DATA (D15-D8)
[P1]
ADDRESS (A13-A1)
+5V
R19 10K
R20 10K
HM6264P-15
U22
U23
TBP18S030
U24
74S32
U1
GO
IAROM
RD
RAMHI
RAMLO
RD [P1.3.5]
WR [P1.5]
RAMBUS [P1]


CONNECTOR
CON1
CON2
82530
U29
1488
1489
U30
U31
DATA (D7-D0)
ADDRESS
292010-83



292010-84

```
Module Addr_dec
Title 'LANHIB Address Decode Logic
       Kiyoshi Nishide    Intel Corp.    March, 1986'

"Declarations

    PAL1                          device          'P16L8';

    A0, A14, A15       pin                        1, 2, 3;
    A16, A17, A18      pin                        4, 5, 6;
    A19, BHE                      pin             7, 8;
    HLDA, S2                      pin             9, 11;
    RAMLO, RAMHI       pin                        18, 17;
    ROMLO, ROMHI       pin                        19, 12;
    ROM                pin        13;
    R104               pin        16;

Equations

    !ROMHI  =  A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROMLO  =  !A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROM  =  A17 & A18 & A19 & (HLDA # S2) & !R104;
    !RAMHI  =  !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & !BHE & (HLDA # S2);
    !RAMLO  =  !A0 & !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & (HLDA # S2);

End Addr_dec
```

PAL Equations

DIP SWITCH SETTINGS FOR VARIOUS OPERATIONS

"1" indicates ON (Switch is closed).
"0" indicates OFF (Switch is open).
"X" indicates Don't Care.

1. To configure the board to Ethernet or Cheapernet:

	SW3 87654321	Comment
Ethernet	XX000000	
Cheapernet	XX111111	Transceiver Cable should not be connected.

2. To run the TSMS program or the Data Link Driver program:

	SW4 87654321	Comment
TSMS Program or Data Link Driver Program	XXXX0001	TSMS program uses the 82530 in Asynchronous Polling mode. Data Link Driver program uses the 825830 in Asynchronous Polling and Vectored Interrupt modes.

3. To select the 2764-20 EPROMs or 27210 EPROM:

	SW3 87654321
2764-20 EPROMs	0XXXXXXX
27210 EPROM	1XXXXXXX

4. Dip Switch Setting Examples:

	SW3 87654321	SW4 87654321
1) To run the TSMS Program from the 2764-20 EPROMs in Cheapernet Configuration	0X111111	XXXX0010
2) To run the TSMS Program from the 2764-20 EPROMs in Ethernet Configuration	0X000000	XXXX0010
3) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Cheapernet Configuration	1X111111	XXXX0001
4) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Ethernet Configuration	1X000000	XXXX0001

5. Dip Switch SW2 programs the number of wait states for the 82586 (see Table 3).

COMPANIES OFFERING WIRE WRAP SERVICES

AUGAT
Interconnection Systems Division

40 Perry Avenue
P.O. Box 1037
Attleboro, MA 02703
(617) 222-2202

100935 South Wilcrest Drive
Houston, TX 77099
(713) 495-3100

Automation Delectronics Corporation

1650 Locust Avenue
Bohemia, NY 11716
(516) 567-7007

dataCon, Inc.

Eastern Division
60 Blanchard Road
Burlington, MA 01803
(617) 273-5800

Mid-Western Division
502 Morse Avenue
Schaumburg, IL 60193
(312) 529-7690

Western Division
20150 Sunburst Street
Chatsworth, CA 91311-6280
(818) 700-0600

South-Western Division
1829 Monetary Lane
Carrollton, TX 75006
(214) 245-6161

European Division
In der Klinge 5
D-7100 Heilbronn, West Germany
(01731) 217 12

DATAWRAP

37 Water Street
Wakefield, MA 01880
(617) 938-8911

Elma/EMS
A Division of Sandberg Industries

Berkshire Industrial Park
Bethel, CT 06801
(203) 797-9711

1851 Reynolds Avenue
Irvine, CA 92714
(714) 261-9473

3042 Scott Boulevard
Santa Clara, CA 95054
(408) 970-8874

WRAPEX Corporation

96 Mill Street
Woonsocket, RI 02895
(401) 769-3805

APPENDIX B
SOFTWARE LISTINGS—TSMS PROGRAM AND LANHIB INITIALIZATION ROUTINE

```
        /****************************************************************************/
        /*                                                                          */
        /*              Traffic Simulator/Monitor Station Program                   */
        /*              for 186/586 High Integration Board and                      */
        /*              iSBC 186/51                                                 */
        /*                                                                          */
        /*              Ver. 1.0                December 17, 1984                   */
        /*                                                                          */
        /*              Kiyoshi Nishide         Intel Corporation                   */
        /*                                                                          */
        /****************************************************************************/

        /* This software can be conditionally compiled to work on the iSBC 186/51 or
           on the LANHIB.  If 'set(SBC18651)' is added to the compiler call statement,
           this source program will be compiled for the iSBC18651.  */

1       tsms:

        do;

2   1   declare main label public;

        /* literals */

        $IF SBC18651

        declare lit             literally 'literally',
                true                    lit '1',
                false                   lit '0',
                forever                 lit 'while 1',
                ISCP$LOC$LO             lit '0FFF0H',
                ISCP$LOC$HI             lit '0',
                SCB$BASE$LO             lit '0',
                SCB$BASE$HI             lit '0',
                CA$PORT                 lit '0C8H',
                BOARD$ADDRESS$BASE      lit '0F0H',
                INT$TYPE$586            lit '20H',
                INT$TYPE$TIMER0         lit '30H',
                INT$CTL$TIMER0          lit '0FF32H',
                INT$7                   lit '27H',
                PIC$MASK$130            lit '0E2H',
                PIC$MASK$186            lit '0FF28H',
                ENABLE$586              lit '0FEH',
                ENABLE$586$186          lit '0EEH',
                PIC$EOI$130             lit '0E0H',
                EOI$CMD0$130            lit '60H',
                EOI$CMD4$130            lit '64H',
                PIC$EOI$186             lit '0FF22H',
                EOI$CMD0$186            lit '0',
                PIC$VTR$186             lit '0FF20H',
```

292010-31

Traffic Simulator/Monitor Station Program

```
                TIMERO$CTL              lit 'OFF56H',
                TIMERO$COUNT            lit 'OFF50H',
                MAX$COUNT$A             lit 'OFF52H',
                CA                      lit '0',
                ESI$PORT                lit '0CBH',
                NO$LOOPBACK             lit '8',
                LOOPBACK                lit '0';

        $ELSE

3   1   declare lit              literally 'literally',
                true                    lit '1',
                false                   lit '0',
                forever                 lit 'while 1',
                ISCP$LOC$LO             lit '03FF8H',
                ISCP$LOC$HI             lit '0',
                SCB$BASE$LO             lit '0',
                SCB$BASE$HI             lit '0',
                CA$PORT                 lit '8000H',
                BOARD$ADDRESS$BASE      lit '8180H',
                INT$TYPE$586            lit '12',
                INT$TYPE$TIMERO         lit '8',
                INT$CTL$TIMERO          lit 'OFF32H',
                PIC$MASK$186            lit 'OFF28H',
                ENABLE$586              lit '0EFH',
                ENABLE$586$186          lit '0EEH',
                PIC$EOI$186             lit 'OFF22H',
                EOI$CMDO$186            lit '12',
                EOI$CMD4$186            lit '8',
                TIMERO$CTL              lit 'OFF56H',
                TIMERO$COUNT            lit 'OFF50H',
                MAX$COUNT$A             lit 'OFF52H',
                CA                      lit '0',
                ESI$PORT                lit '8100H',
                NO$LOOPBACK             lit '1',
                LOOPBACK                lit '0';

        $ENDIF

        $IF NOT SBC18651

        /* System Configuration Pointer */

4   1   declare scp structure
                (
                sysbus byte,
                unused (5) byte,
                iscp$addr$lo word,
                iscp$addr$hi word
                )
                at (OFFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);

        $ENDIF

        /* Intermediate System Configuration Pointer */
```

292010-32

Traffic Simulator/Monitor Station Program (Continued)

```
5   1       declare iscp$ptr pointer,
                    iscp based iscp$ptr structure
                    (
                    busy byte,       /* set to 1 by CPU before its first CA to 586,
                                        cleared by 586 after reading info from it */
                    unused byte,     /* unused */
                    scb$o word,      /* offset of system control block */
                    scb$b (2) word /* base of system control block */
                    );

            /* System Control Block */

6   1       declare scb structure
                    (
                    status word,        /* cause(s) of interrupt, CU state, RU state */
                    cmd word,           /* int acks, CU cmd, RESET bit, RU cmd */
                    cbl$offset word,    /* offset of first command block in CBL */
                    rpa$offset word,    /* offset of first packet descriptor in RPA */
                    crc$errs word,      /* crc error encounterd so far */
                    aln$errs word,      /* alignment errors */
                    rsc$errs word,      /* no resources */
                    ovrn$errs word      /* overrun errors */
                    );

            /* 82586 Action Commands */

            /* NOP */

7   1       declare nop structure
                    (
                    status word,
                    cmd word,
                    link$offset word
                    );

            /* Individual Address Setup */

8   1       declare ia$setup structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    ia$address (6) byte
                    );

            /* Configure */

9   1       declare configure structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    byte$cnt byte,
                    info (11) byte
                    );
```

292010–33

Traffic Simulator/Monitor Station Program (Continued)

```
                /* Multicast Address Setup */

10    1         declare mc$setup structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        mc$byte$count word,
                        mc$address (48) byte   /* only 8 MC addresses are allowed */
                        );


                /* Transmit */

                /* This transmit command is made of one transmit buffer descriptor and one
                   1518 bytes long buffer. */

11    1         declare transmit structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        bd$offset word,
                        dest$adr (6) byte,
                        type word
                        );

                /* Transmit Buffer Descriptor */

12    1         declare tbd structure
                        (
                        act$count word,
                        link$offset word,
                        ad0 word,
                        ad1 word
                        );

                /* Transmit Buffer */

13    1         declare tx$buffer (1518) byte;


                /* TDR */

14    1         declare tdr structure
                        (
                        status word,
                        cmd word,
                        link$offset word,
                        result word
                        );

                /* Diagnose */

15    1         declare diagnose structure
                        (
```

292010–34

Traffic Simulator/Monitor Station Program (Continued)

```
                status word,
                cmd word,
                link$offset word
                );

          /* Dump Status */

16   1    declare dump structure
                (
                status word,
                cmd word,
                link$offset word,
                buff$ptr word
                );

          /* Dump Area */

17   1    declare dump$area (170) byte;

          /* Frame Descriptor */

          /* Receive frame area is made of 5 RFDs, 5 RBDs, and 5 1514 bytes long
             buffers. */

18   1    declare rfd (5) structure
                (
                status word,
                el$s word,
                link$offset word,
                bd$offset word,
                dest$adr (3) word,
                src$adr (3) word,
                type word
                );

          /* Receive Buffer Descriptor */

19   1    declare rbd (5) structure
                (
                act$count word,
                next$bd$link word,
                ad0 word,
                ad1 word,
                size word
                );

          /* Receive Buffer */

20   1    declare rbuf (5) structure
                (buffer (1514) byte);

          /* global variables */

21   1    declare status word,              /* UART status */
```

292010-35

Traffic Simulator/Monitor Station Program (Continued)

```
                  actual word,              /* actual number of chars UART transferred */
                  c$buf (80) byte,          /* buffer for a line of chars */
                  dhex byte,                /* number base switch */
                  ch byte at (@c$buf),
                  char$count byte,
                  receive$count dword,      /* counter for received frames */
                  count dword,              /* counter for transmitted frames */
                  preamble word,            /* preamble length in word */
                  address$length byte,      /* address length in byte */
                  ad$loc byte,              /* address location control of 82586 */
                  crc byte,                 /* crc length */
                  goback byte,              /* if set, go back to Continuous Mode */
                  reset byte,               /* reset flag */
                  delay word,               /* delay conunt for tranmission delay */
                  cur$cb$offset word,       /* offset of current command block */
                  current$frame byte,       /* offset of frame descriptor just used */
                  no$transmission byte,
                  stop$count dword,         /* transmit terminal frame count */
                  stop byte,
                  mc$count byte,
                  z byte,
                  y byte;

           /* external procedures */

22   1     read: procedure (a, b, c, d, e) external;
23   2     declare (a, c) word,
                  (b, d, e) pointer;
24   2     end read;


25   1     write: procedure (a, b, c, d) external;
26   2     declare (a, c) word,
                  (b, d) pointer;
27   2     end write;


28   1     csts: procedure byte external;
29   2     end csts;


           /* utility procedures */

30   1     offset: procedure (ptr) word;

           /* This procedure takes a pointer variable (selector:offset), caluculates an
              absolute address, subtracts the 82586 SCB offset from the absolute address,
              and then returns the result as an offset value for the 82586.  */

31   2     declare (ptr, ptr$loc) pointer,
                  base586 dword,
                  w based ptr$loc (2) word;

32   2        ptr$loc = @ptr;

              /* 82586 SCB Base Address (20-bit wide in this 186 based system) */
```

292010-36

Traffic Simulator/Monitor Station Program (Continued)

```
33   2          base586 = (shl(double (iscp scb$b(1)), 16) and 000F0000H) + iscp.scb$b(0);
34   2          return low((shl(double (w(1)), 4) + w(0)) - base586);

35   2      end offset;


36   1      writeln: procedure (a, b, c, d);

            /* This procedure writes a line and put a CR/LF at the end. */

37   2      declare (a, c) word,
                    (b, d) pointer;

38   2          call write(a, b, c, d);
39   2          call write(0, @(0DH, 0AH), 2, @status);

40   2      end writeln;


41   1      cr$lf: procedure;

            /* This procedure writes a CR/LF. */

42   2          call write (0, @(0DH, 0AH), 2, @status);

43   2      end cr$lf;


44   1      pause: procedure;

            /* This procedure breaks a program flow, and waits for a char to be typed. */

45   2          call write(0, @(0DH, 0AH, 'Hit <CR> to countinue'), 23, @status);
46   2          call read(1, @c$buf, 80, @actual, @status);
47   2          call cr$lf;

48   2      end pause;


49   1      skip: procedure byte;

            /* This procedure skips all leading blank characters and returns the first
               non-blank character. */

50   2      declare i byte;

51   2          i = 0;
52   2          do while (c$buf(i) = ' ');
53   3              i = i + 1;
54   3          end;
55   2          return i;

56   2      end skip;


57   1      read$char: procedure byte;
```

292010–37

Traffic Simulator/Monitor Station Program (Continued)

```
                /* This procedure reads a line and returns ther first non-blank character.  */

58    2         declare i word;

59    2             call read(1, @c$buf, 80, @actual, @status);
60    2             i = skip;
61    2             return(c$buf(i));

62    2         end read$char;


63    1         read$bit: procedure byte;

                /* This procedure reads a bit and returns the value.  */

64    2         declare b byte;

65    2             do forever;
66    3                 b = read$char;
67    3                     if b = '1' then return 1;
69    3                 else
70    3                     if b = '0' then return 0;
71    3                 else
                            call write(0, @(' Enter a 0 or 1 ==> '), 20, @status);
72    3             end;

73    2         end read$bit;


74    1         yes: procedure byte;

                /* This procedure reads a character and determines if it is a Y(y) or N(n).  */

75    2         declare b byte;

76    2             do forever;
77    3                 b = read$char;
78    3                 if (b = 'Y') or (b = 'y') then return true;
80    3                 else
81    3                     if (b = 'N') or (b = 'n') then return false;
82    3                 else
                            call write(0, @(0DH, 0AH, ' Enter a Y or N ==> '), 22, @status);
83    3             end;

84    2         end yes;


85    1         char$to$int: procedure (c) byte;

                /* This procedure converts a byte of ASCII integer to an integer.  */

86    2         declare c byte;

87    2             if ('0' <= c) and (c <= '9') then return (c - 30H);
89    2             else
90    2                 if('A' <= c) and (c <= 'F') then return (c - 37H);
91    2                 else
```

292010-38

Traffic Simulator/Monitor Station Program (Continued)

```
 92   2                  if ('a' <= c) and (c <= 'f') then return (c - 57H);
 93   2                  else return OFFH;

 94   2        end char$to$int;


 95   1        int$to$asci: procedure (value, base, ld, bufadr, width);

               /* This procedure converts an interger < OFFFFFFFFH to an array of ASCII
                  codes.
                  Input variables are:   value = integer to be converted,
                                         base = number base to be used for conversion,
                                         ld = leading character to be filled in,
                                         bufadr = buffer address of the array,
                                         width = size of array. */

 96   2        declare value dword,
                       bufadr pointer,
                       (i, j, base, ld, width) byte,
                       chars based bufadr (1) byte;

 97   2            do i = 1 to width;
 98   3                j = value mod base;
 99   3                if j < 10 then chars (width - i) = j + 30H;
101   3                else chars (width - i) = j + 37H;
102   3                value = value / base;
103   3            end;
104   2            i = 0;
105   2            do while chars (i) = '0' and i < width - 1;
106   3                chars (i) = ld;
107   3                i = i + 1;
108   3            end;
109   2            char$count = width - i;

110   2        end int$to$asci;


111   1        out$word: procedure (w$ptr, distance);

               /* An integer at (selector of w$ptr):(offset of w$ptr + distance) is printed
                  as a 4 digit hexadecimal number.  */

112   2        declare chars(4) byte,
                       w$ptr pointer,
                       distance byte,
                       w based w$ptr (1) word;

113   2            call int$to$asci(w(distance), 16, '0', @chars(0), 4);
114   2            call write(0, @chars(0), 4, @status);

115   2        end out$word;


116   1        write$int: procedure(dw, t);

               /* An integer (dw) is printed in hexadecimal (t = 1) or in decimal (t = 0). */
```

292010-39

Traffic Simulator/Monitor Station Program (Continued)

```
117   2        declare dw dword,
                      chars (10) byte,
                      t byte;

118   2           if t then
119   2           do;
120   3               call int$to$asci(dw, 16, 0, @chars(0), 8);
121   3               call write(0, @chars(8-char$count), char$count, @status);
122   3           end;
123   2           else
                  do;
124   3               call int$to$asci(dw, 10, 0, @chars(0), 10);
125   3               call write(0, @chars(10-char$count), char$count, @status);
126   3           end;

127   2        end write$int;


128   1        out$dec$hex: procedure(dw);

               /* This procedure prints an integer in decimal and hexadecimal. */

129   2        declare dw dword;

130   2           call write$int(dw, 0);
131   2           call write(0, @(' ('), 2, @status);
132   2           call write$int(dw, 1);
133   2           call write(0, @('H)'), 2, @status);

134   2        end out$dec$hex;


135   1        write$offset: procedure(w$ptr);

               /* This procedure takes a pointer variable, converts it to a 82586 type offset,
                  and prints it in hexadecimal.  */

136   2        declare w$ptr pointer,
                      w word;

137   2           call write(0, @(' at '), 4, @status);
138   2           w = offset(w$ptr);
139   2           call out$word(@w, 0);
140   2           call write(0, @('  '), 2, @status);

141   2        end write$offset;


142   1        write$address: procedure (ptr);

               /* This procedure takes a pointer variable and prints it in the
                  'selector:offset' format.  */

143   2        declare (ptr, ptr$loc)  pointer,
                      w based ptr$loc (2) word;

144   2           ptr$loc = @ptr;
```

292010-40

Traffic Simulator/Monitor Station Program (Continued)

```
145   2           call out$word(@w(1), 0);
146   2           call write(0, @(':'), 1, @status);
147   2           call out$word(@w(0), 0);
148   2           call write(0, @(' '), 1, @status);

149   2       end write$address;


150   1       print$wds: procedure(w$ptr, no$words);

              /* This procedure prints no$words number of words starting at w$ptr.   */

151   2       declare w$ptr pointer,
                      (i, no$words) byte;

152   2           if no$words <> 0 then
153   2           do;
154   3               call cr$lf;
155   3               do i = 0 to no$words - 1;
156   4                   call out$word(w$ptr, i);
157   4                   if i = 0 then
158   4                       call write$offset(w$ptr);
159   4                   call cr$lf;
160   4               end;
161   3           end;

162   2       end print$wds;


163   1       print$str: procedure (str$ptr, len);

              /* This procedure prints len number of bytes starting at str$ptr. */

164   2       declare (len, i) byte,
                      chars (2) byte,
                      str$ptr pointer,
                      str based str$ptr (1) byte;

165   2           if len <> 0 then
166   2           do i = 0 to (len - 1);
167   3               call int$to$asci(str(i), 16, '0', @chars(0), 2);
168   3               call write(0, @chars(0), 2, @status);
169   3               call write(0, @('  '), 2, @status);
170   3           end;
171   2           call cr$lf;

172   2       end print$str;


173   1       print$buff: procedure (ptr, cnt);

              /* This procedure prints cnt number of buffer contents starting at ptr.  */

174   2       declare ptr pointer,
                      bt based ptr (1) byte,
                      (i, j) byte,
                      cnt word;
```

292010–41

Traffic Simulator/Monitor Station Program (Continued)

```
175  2          if cnt > 16 then
176  2          do;
177  3              i = shr(cnt, 4) - 1;
178  3              do j = 0 to i;
179  4                  call write$address(@bt(16*j));
180  4                  call print$str(@bt(16*j), 16);
181  4                  if (j = 20) or (j = 40) or (j = 60) or (j = 80) then
182  4                  call pause;
183  4              end;
184  3              i = i + 1;
185  3              if cnt-16*i <> 0 then call write$address(@bt(16*i));
187  3              call print$str(@bt(16*i), cnt-16*i);
188  3          end;
189  2          else
                do;
190  3              call write$address(@bt(0));
191  3              call print$str(@bt(0), cnt);
192  3          end;

193  2      end print$buff;


194  1      read$int: procedure (limit) dword;

            /* This procedure reads integer characters and forms an integer.  If the
               integer is bigger than 'limit' or an overflow error is encounterred, then
               an error message is printed.  */

195  2      declare (wd, wh, limit) dword,
                    (i, j, k, done, hex, dover, hover) byte;

196  2          do forever;
197  3              call read(1, @c$buf, 80, @actual, @status);
198  3              i, k = skip;
199  3              hex, done, dover, hover = false;
200  3              wd, wh = 0;
201  3              j = char$to$int(c$buf(i));
202  3              do while j <= 15;
203  4                  if j > 9 then hex = true;
205  4                  if not dover then
206  4                      if wd > 429496729 then dover = true;
208  4                      else if (wd = 429496729) and (j > 5) then dover = true;
210  4                  wd = wd*10 + j;
211  4                  if not hover then if wh > 0FFFFFFFH then hover = true;
214  4                  wh = wh*16 + j;
215  4                  i = i + 1;
216  4                  j = char$to$int(c$buf(i));
217  4              end;
218  3              if ((c$buf(i) <> 'H') and (c$buf(i) <> 'h') and (c$buf(i) <> 0DH) and
                        (c$buf(i) <> 0AH) and (c$buf(i) <> ' ')) or (i = k) then
219  3              call writeln(0, @(0DH, 0AH, ' Illegal character'), 20, @status);
220  3              else
                    do;
221  4                  if (c$buf(i) = 'H') or (c$buf(i) = 'h') then hex = true;
223  4                  if hex then
```

292010-42

Traffic Simulator/Monitor Station Program (Continued)

```
224   4                  do;
225   5                      if not hover and (wh <= limit) then return wh;
227   5                  end;
228   4                  else
229   4                      if not dover and (wd <= limit) then return wd;
230   4                  call writeln(0, @(ODH, OAH, ' The number is too big.'), 25,
                                                                          @status);
231   4                  call write(0, @(' It has to be less than or equal to '), 36,
                                                                          @status);
232   4                  call out$dec$hex(limit);
233   4                  call writeln(0, @('.'), 1, @status);
234   4              end;
235   3              call write(0, @(' Enter a number ==> '), 20, @status);
236   3          end;

237   2      end read$int;


238   1      put$address: procedure(where);

      /* This procedure puts an address typed in hexadecimal to the specified
         location 'where'. */

239   2      declare where pointer,
                     (i, j, m, err) byte,
                     addr based where (1) byte;

240   2          do forever;
241   3              err = false;
242   3              call read(1, @c$buf, 80, @actual, @status);
243   3              i = skip;
244   3              m = address$length;
245   3              do while (m <> 0) and not err;
246   4                  j = char$to$int(c$buf(i));
247   4                  if j = OFFH then err = true;
249   4                  else
                          do;
250   5                      addr(m-1) = shl(j, 4);
251   5                      j = char$to$int(c$buf(i+1));
252   5                      if j = OFFH then err = true;
254   5                      else addr(m-1) = addr(m-1) or j;
255   5                  end;
256   4                  i = i + 2;
257   4                  m = m - 1;
258   4              end;
259   3              if not err then
260   3              do;
261   4                  m = c$buf(i);
262   4                  if (m = ODH) or (m = OAH) or (m = 'h') or (m = 'H') or (m = ' ')
263   4                      then return;
264   4              end;
265   3              call writeln(0, @(ODH, OAH, ' Illegal character'), 20, @status);
266   3              call write(0, @(' Enter an address in Hex ==> '), 29, @status);
267   3          end;

268   2      end put$address;
```

292010-43

Traffic Simulator/Monitor Station Program (Continued)

```
269   1      percent: procedure;

             /* This procedure calculates and prints a network percent load generated
                by this station.  The equation used in this procedure was obtained
                from actual measurements.  */

270   2      declare i word,
                     (j, k) dword,
                     pcent (3) byte;

271   2         j = (tbd.act$count and 3FFFH)*8;
272   2         if not ad$loc then k = (2*address$length + 2 + crc + preamble)*8;
274   2         else k = (crc + preamble)*8;
275   2         if delay <> 0 then

             $IF NOT SBC18651

276   2         i = low((1000*(j + k))/(1805 + k + 5*double(delay) + j));

             $ELSE

                i = low((1000*(j + k))/(2021 + k + 5*double(delay) + j));

             $ENDIF

277   2         else

             $IF NOT SBC18651

                i = low((1000*(j + k))/(1810 + k + j));

             $ELSE

                i = low((1000*(j + k))/(2026 + k + j));

             $ENDIF

278   2         call int$to$asci(i, 10, 0, @pcent(0), 3);
279   2         call write(0, @pcent(0), 2, @status);
280   2         call write(0, @('.'), 1, @status);
281   2         call write(0, @pcent(2), 1, @status);
282   2         call writeln(0, @(' %'), 2, @status);

283   2      end percent;


284   1      print$network$addr: procedure (ptr);

             /* This station's address is printed with its least significant bit
                in the most right position.  */

285   2      declare ptr pointer,
                     addr based ptr (1) byte,
                     char (6) byte,
                     i byte;
```

292010-44

Traffic Simulator/Monitor Station Program (Continued)

```
        do i = 1 to address$length;
            char(i-1) = addr(address$length-i);
        end;
        call print$str(@char(0), address$length);

    end print$network$addr;


    print$parameters: procedure;

    /* This procedure prints transmission parameters. */

    declare w dword,
            stgs (6) byte;

        call write(0, @(' Destination Address: '), 22, @status);
        if not ad$loc then
            call print$network$addr(@transmit.dest$adr(0));
        else
            call print$network$addr(@tx$buffer(0));
        if not ad$loc then
        w =  (tbd.act$count and 3FFFH) + address$length * 2 + 2 + crc;
        else w = (tbd.act$count and 3FFFH) + crc;
        call write(0, @(' Frame Length: '), 15, @status);
        call write$int(w, 0);
        call writeln(0, @(' bytes'), 6, @status);
        call write(0, @(' Time Interval between Transmit Frames: '), 40, @status);
        if delay <> 0 then
        do;
    $IF NOT SBC18651

            w = 1810 + (double(delay) - 1) * 5;

    $ELSE

            w = 2026 + (double(delay) - 1) * 5;

    $ENDIF

            call int$to$asci(w, 10, 0, @stgs, 6);
            if w >= 10000 then
            do;
                call write(0, @stgs(0), 2, @status);
                call write(0, @('.'), 1, @status);
                call write(0, @stgs(2), 2, @status);
                call writeln(0, @(' miliseconds'), 12, @status);
            end;
            else
            do;
                call write(0, @stgs(0), 5, @status);
                call write(0, @('.'), 1, @status);
                call write(0, @stgs(5), 1, @status);
                call writeln(0, @(' microseconds'), 13, @status);
            end;
        end;
        else
```

292010-45

Traffic Simulator/Monitor Station Program (Continued)

```
             $IF NOT SBC18651
                 call writeln(0, @(' 159.4 microseconds'), 19, @status);
             $ELSE
                 call writeln(0, @(' 172.8 microseconds'), 19, @status);
             $ENDIF
323   2          call write(0, @(' Network Percent Load generated by this station:  '), 49,
                                                                              @status);
324   2          call percent;
325   2          call write(0, @(' Transmit Frame Terminal Count:  '), 32, @status);
326   2          if stop then call write$int(stop$count, dhex);
328   2              else call write(0, @('Not Defined'), 11, @status);
329   2          call cr$lf;

330   2      end print$parameters;


331   1      print$scb: procedure;

             /* prints the SCB */

332   2          call writeln(0, @(0DH, 0AH, '*** System Control Block ***'), 30, @status);
333   2          call print$wds(@scb.status, 8);

334   2      end print$scb;



335   1      wait$scb: procedure;

             /* This procedure provids a wait loop for the SCB command word to
                become cleared.   */

336   2      declare i word;

337   2          i = 0;
338   2          do while (scb.cmd <> 0) and (i < 8000H);
339   3              i = i + 1;
340   3          end;
341   2          if scb.cmd <> 0 then
342   2              do;
343   3                  call write(0, @(0DH, 0AH, ' Wait Time = '), 15, @status);
344   3                  call write$int(i, 0);
345   3                  call cr$lf;
346   3              end;

347   2      end wait$scb;


348   1      start$timer0: procedure;

             /* 80186 timer0 is started.  */
```

292010-46

Traffic Simulator/Monitor Station Program (Continued)

```
349   2          output(TIMERO$CTL) = OE000H;

350   2     end start$timer0;


351   1     isr: procedure interrupt INT$TYPE$586 reentrant;

            /* interrupt service routine for 82586 interrupt */

352   2     declare i byte;

                /* Enable 82586 Interrupt */

            $IF SBC18651

                output (PIC$EOI$130) = EOI$CMDO$130;
                enable;

            $ELSE

353   2         output (PIC$EOI$186) = EOI$CMDO$186;
354   2         enable;

            $ENDIF

                /* Frame Received Interrupt has the highest priority */

355   2         if (scb.status and 4000H) = 4000H then
356   2         do;
357   3             disable;
358   3             scb.cmd = 4000H;
359   3             output (CA$PORT) = CA;
360   3             call wait$scb;
361   3             if rfd(current$frame).status = 0A000H then
362   3             do;
363   4                 receive$count = receive$count + 1;
364   4                 current$frame = current$frame + 1;
365   4                 if current$frame = 5 then current$frame = 0;
367   4             end;
368   3             return;
369   3         end;

370   2         if (scb.status and 2000H) = 2000H then
371   2         do;
372   3             disable;
373   3             scb.cmd = 2000H;
374   3             output(CA$PORT) = CA;
375   3             call wait$scb;
376   3             enable;
377   3             if (transmit.status and 0A000H) = 0A000H then
378   3             do;
379   4                 count = count + 1;
380   4                 if (stop and (count = stop$count)) then return;
382   4                 else
                        do;
```

292010–47

Traffic Simulator/Monitor Station Program (Continued)

```
383   5                 transmit.status = 0;
384   5                 if delay = 0 then
385   5                 do;
386   6                     disable;
387   6                     scb.cmd = 0100H;
388   6                     output(CA$PORT) = CA;
389   6                     call wait$scb;
390   6                     return;
391   6                 end;
392   5                 else
                        do;
393   6                     call start$timer0;
394   6                     return;
395   6                 end;
396   5             end;
397   4         end;
398   3         if (transmit.status and 0020H) = 0020H then
399   3         do;
400   4             transmit.status = 0;
401   4             disable;
402   4             scb.cmd = 0100H;
403   4             output (CA$PORT) = CA;
404   4             call wait$scb;
405   4             return;
406   4         end;
407   3         if (transmit.status and 0400H) = 0400H then
408   3         do;
409   4             call write(0, @(0DH, ' No Carrier Sense!', 0DH), 20, @status);
410   4             transmit.status = 0;
411   4             disable;
412   4             scb.cmd = 0100H;
413   4             output (CA$PORT) = CA;
414   4             call wait$scb;
415   4             return;
416   4         end;
417   3         if (transmit.status and 0200H) = 0200H then
418   3         do;
419   4             call write(0, @(0DH, ' Lost Clear to Send!', 0DH), 22, @status);
420   4             transmit.status = 0;
421   4             disable;
422   4             scb.cmd = 0100H;
423   4             output (CA$PORT) = CA;
424   4             call wait$scb;
425   4             return;
426   4         end;
427   3         if (transmit.status and 0100H) = 0100H then
428   3         do;
429   4             call write(0, @(0DH, ' DMA Underrun!', 0DH), 16, @status);
430   4             transmit.status = 0;
431   4             disable;
432   4             scb.cmd = 0100H;
433   4             output (CA$PORT) = CA;
434   4             call wait$scb;
435   4             return;
436   4         end;
437   3     end;
438   2     if (scb.status and 8000H) = 8000H then
```

292010-48

Traffic Simulator/Monitor Station Program (Continued)

```
439   2        do;
440   3            disable;
441   3            scb.cmd = 8000H;
442   3            output (CA$PORT) = CA;
443   3            call wait$scb;
444   3        end;
445   2        if (scb.status and 1000H) = 1000H then
446   2        do;
447   3            disable;
448   3            scb.cmd = 1000H;
449   3            output (CA$PORT) = CA;
450   3            call wait$scb;
451   3            call write(0, @(ODH, ' Receive Unit became not ready. ', ODH), 33,
                                                                           @status);
452   3        end;

453   2        if reset then
454   2        do;
455   3            if iscp.busy then
456   3            do;
457   4                call writeln(0, @(ODH, OAH, ' Reset failed. '), 16, @status);
458   4                disable;
459   4                scb.cmd = 0080H;
460   4                output (CA$PORT) = CA;
461   4                call wait$scb;
462   4                output (CA$PORT) = CA;
463   4                call writeln(0, @(' Software Reset Executed!'), 25, @status);
464   4            end;
465   3            else reset = false;
466   3        end;

467   2    end isr;



468   1    tx$isr: procedure interrupt INT$TYPE$TIMERO;

           /* interrupt survice routine for 80186 timer interrupt*/

469   2        scb.cmd = 0100H;
470   2        output(CA$PORT) = CA;
471   2        call wait$scb;

           $IF SBC18651

               output(PIC$EOI$130) = EOI$CMD4$130;
               enable;
               output(PIC$EOI$186) = EOI$CMD0$186;

           $ELSE

472   2        output(PIC$EOI$186) = EOI$CMD4$186;

           $ENDIF

473   2    end tx$isr;
```

292010-49

Traffic Simulator/Monitor Station Program (Continued)

```
                $IF SBC18651

                isr$7: procedure interrupt INT$7;

                /* The 80130 generates an interrupt 7 if the original interrupt is not
                   active any more when the first interrupt acknowledge is received.  */

                    call write(0, @(0DH, 'Interrupt 7', 0DH), 13, @status);

                end isr$7;

                $ENDIF


474     1       read$byte: procedure (k) byte;
475     2       declare k word;

476     2           call write(0, @(0DH, 0AH, ' Enter byte '), 14, @status);
477     2           call out$dec$hex(k);
478     2           call write(0, @(' ==> '), 5, @status);
479     2           return read$int(0FFH);

480     2       end read$byte;


481     1       init$186$timer0: procedure;

                /* This procedure initializes the 80186 timer 0.   */

482     2       declare i byte;

                $IF SBC18651

                    output(INT$CTL$TIMER0) = 8;
                    call write(0, @(0DH, 0AH, ' Enter a delay count ==> '), 27, @status);
                    delay = read$int(0FFFFH);
                    if (delay < 100) and (delay <> 0) then
                    do;
                        call cr$lf;
                        call cr$lf;
                        call loop$char(35, '*');
                        call write(0, @(' WARNING '), 9, @status);
                        call loop$char(35, '*');
                        call writeln(0, @(0DH, 0AH, 'A delay count between 0 and 100 may be very ',
                                'dangerous when this station starts'), 80, @status);
                        call writeln(0, @('to receive many frames separated only by the ',
                                'IFS period (9.6 microseconds). '), 75, @status);
                        call writeln(0, @('If this station never receives a frame, then ',
                                         'ignore this warning. '), 65, @status);
                        call loop$char(79, '*');
                    end;
                    output(MAX$COUNT$A) = delay;
                    call cr$lf;
                    output(PIC$MASK$186) = 3EH;
```

292010-50

Traffic Simulator/Monitor Station Program (Continued)

```
                $ElSE

483  2              output(INT$CTL$TIMERO) = OCH;
484  2              call write(O, @(ODH, OAH, ' Enter a delay count ==> '), 27, @status);
485  2              delay = read$int(OFFFFH);
486  2              output(MAX$COUNT$A) = delay;
487  2              call cr$lf;
488  2              output(PIC$MASK$186) = ENABLE$586$186;

                $ENDIF

489  2          end init$186$timer0;


490  1          setup$ia$parameters: procedure;
491  2          declare i byte;

492  2              call write(O, @(ODH, OAH, ' Configure the 586 with the prewired'
                                                        ,' board address ==> '), 57, @status);
493  2              if yes then
494  2              do i = 0 to address$length - 1;
495  3                  ia$setup.ia$address(i) = input(BOARD$ADDRESS$BASE + 10 - 2 * i);
496  3              end;
497  2              else
                    do;
498  3                  call write(O, @(ODH, OAH, ' Enter this station''s address',
                                                              ' in Hex ==> '),43, @status);
499  3                  call put$address(@ia$setup.ia$address(O));
500  3              end;

501  2          end setup$ia$parameters;


502  1          setup$mc$parameters: procedure;
503  2          declare (j, k, done) byte;

504  2              j = 0;
505  2              call writeln(O, @(ODH, OAH, ' You can enter up to 8 Multicast Addresses. '),
                                                                                 45, @status);
506  2              done = false;
507  2              call write(O, @(' Would you like to enter a Multicast Address?',
                                                            ' (Y or N) ==> '), 59, @status);
508  2              do while not done;
509  3                  if yes then
510  3                  do;
511  4                      k = j * address$length;
512  4                      j = j + 1;
513  4                      call cr$lf;
514  4                      if j = 9 then
515  4                      do;
516  5                          call write(O, @(' You already entered 8 Multicast addresses. '),
                                                                              43, @status);
517  5                          done = true;
518  5                      end;
519  4                      else
                            do;
520  5                          call write(O, @(' Enter a Multicast Address ==> '), 31, @status);
```

292010-51

Traffic Simulator/Monitor Station Program (Continued)

```
521  5                  call put$address(@mc$setup.mc$address(k));
522  5                  call write(0, @(0DH, 0AH, ' More Multicast Addresses?',
                                             ' (Y or N) ==> '), 42, @status);
523  5              end;
524  4          end;
525  3          else done = true;
526  3      end;
527  2      if j = 9 then j = j - 1;
529  2      mc$count = address$length * j;
530  2      mc$setup.mc$byte$count = mc$count;
531  2      call write(0, @(0DH, 0AH, ' You entered '), 15, @status);
532  2      call write$int(j, 0);
533  2      call writeln(0, @(' Multicast Address(es).'), 23, @status);

534  2  end setup$mc$parameters;


535  1  setup$configure$parameters: procedure;
536  2  declare (k, j) byte;

537  2      configure.byte$cnt = 11;
538  2      configure.info(0) = 8;
539  2      configure.info(1) = 0;
540  2      configure.info(2) = 26H;
541  2      configure.info(3) = 0;
542  2      configure.info(4) = 96;
543  2      configure.info(5) = 0;
544  2      configure.info(6) = 0F2H;
545  2      configure.info(7) = 0;
546  2      configure.info(8) = 0;
547  2      configure.info(9) = 64;
548  2      j = 0;
549  2      call write(0, @(0DH, 0AH, ' Configure command is set up for default',
                ' values.', 0DH, 0AH, ' Do you want to change any bytes?',
                                             ' (Y or N) ==> '), 99, @status);
550  2      do while yes;
551  3          do while j = 0;
552  4              call write(0, @(0DH, 0AH, ' Enter byte number (1 - 11) ==> '), 34,
                                                                       @status);
553  4              j = read$int(11);
554  4              if j = 0 then
555  4              call write(0, @(0DH, 0AH, ' Illegal byte number'), 22, @status);
556  4          end;
557  3          if j = 1 then configure.byte$cnt = read$byte(j);
559  3              else configure.info(j - 2) = read$byte(j);
560  3          j = 0;
561  3          call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '), 32,
                                                                       @status);
562  3      end;
563  2      preamble = shl(1, shr((configure.info (2) and 30H), 4)+1);
564  2      address$length = configure.info(2) and 07H;
565  2      if address$length = 7 then address$length = 0;
567  2      ad$loc = shr((configure.info(2) and 08H), 3);
568  2      if shr((configure.info(7) and 20H), 5) then crc = 2; else crc = 4;
571  2      if shr((configure.info(7) and 10H), 4) then crc = 0;

573  2  end setup$configure$parameters;
```

292010-52

Traffic Simulator/Monitor Station Program (Continued)

```
574   1     setup$tx$parameters: procedure;
575   2     declare (size, i) word;

576   2         do forever;
577   3             no$transmission = false;
578   3             transmit.bd$offset = offset (@tbd.act$count);
579   3             if not ad$loc then
580   3             do;
581   4                 call write(0, @(0DH, 0AH,
                          ' Enter a destination address in Hex ==> '), 42, @status);
582   4                 call put$address(@transmit.dest$adr(0));
583   4             end;
584   3             else call writeln(0, @(' 82586 is configured to pick up DA, IA, ',
                                   ' and TYPE from TX buffer. '), 64, @status);
585   3             call cr$lf;
586   3             if not ad$loc then
587   3             do;
588   4                 call write(0, @(0DH, 0AH, ' Enter TYPE ==> '), 18, @status);
589   4                 transmit.type = read$int(0FFFFH);
590   4             end;
591   3             call writeln(0, @(0DH, 0AH,' How many bytes of transmit data?'), 35,
                                                                        @status);
592   3             call write(0, @(' Enter a number ==> '), 20, @status);
593   3             size = read$int(1518);
594   3             tbd.act$count = size or 8000H;
595   3             if size <> 0 then
596   3             do;
597   4                 tbd.link$offset = 0FFFFH;
598   4                 tbd.ad0 = offset (@tx$buffer(0));
599   4                 tbd.ad1 = 0;
600   4                 do i = 0 to 1517;
601   5                     tx$buffer(i) = i;
602   5                 end;
603   4                 call writeln(0,
                        @(0DH, 0AH, ' Transmit Data is continuous numbers (0, 1, 2, 3, ',
                                                              ' ... )'), 57, @status);
604   4                 call write(0, @(' Change any data bytes? (Y or N) ==> '), 37,
                                                                        @status);
605   4                 do while yes;
606   5                     call write(0, @(0DH, 0AH, ' Enter a byte number ==> '),
                                                                    27, @status);
607   5                     i = read$int(size);
608   5                     call write(0, @(0DH, 0AH, ' Byte '), 8, @status);
609   5                     call out$dec$hex(i);
610   5                     call write(0, @(' currently contains '), 20, @status);
611   5                     call out$dec$hex(tx$buffer(i));
612   5                     call write(0, @('.'), 1, @status);
613   5                     tx$buffer(i) = read$byte(i);
614   5                     call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '),
                                                                    32, @status);
615   5                 end;
616   4             end;
617   3             else
                    transmit.bd$offset = 0FFFFH;
618   3             call cr$lf;
```

292010-53

Traffic Simulator/Monitor Station Program (Continued)

```
619   3              call init$186$timer0;
620   3              call write(0, @(0DH, 0AH, ' Setup a transmit terminal count?',
                                                  ' (Y or N) ==> '), 49, @status);
621   3              if yes then
622   3              do;
623   4                  stop = true;
624   4                  call write (0, @(0DH, 0AH, ' Enter a transmit',
                                              ' terminal count ==> '), 39, @status);
625   4                  stop$count = read$int(0FFFFFFFFH);
626   4              end;
627   3              else stop = false;
628   3              call cr$lf;
629   3              call cr$lf;
630   3              call print$parameters;
631   3              call write(0, @(0DH, 0AH, ' Good enough? (Y or N) ==> '), 29,
                                                                          @status);
632   3              if yes then return;
634   3          end;

635   2      end setup$tx$parameters;


636   1      loop$char: procedure (i, j);
637   2      declare (i, j, k) byte;

638   2          do k = 1 to i;
639   3              call write(0, @j, 1, @status);
640   3          end;

641   2      end loop$char;


642   1      init: procedure;

643   2      declare  i byte;

644   2          call cr$lf;
645   2          call loop$char(13, 0AH);
646   2          call loop$char(15, ' ');
647   2          call writeln(0, @('TRAFFIC SIMULATOR AND MONITOR',
                                        ' STATION PROGRAM '), 46, @status);
648   2          call loop$char(7, 0AH);
649   2          call writeln(0, @(0DH, 0AH, ' Initialization begun'), 23, @status);
650   2          call cr$lf;
651   2          reset = true;
652   2          cur$cb$offset = 0FFFFH;
653   2          output(ESI$PORT) = NO$LOOPBACK;
654   2          output(ESI$PORT) = LOOPBACK;
655   2          dhex = false;

             /* set up interrupt logic */

656   2          call set$interrupt(INT$TYPE$586, isr);
657   2          call set$interrupt(INT$TYPE$TIMER0, tx$isr);

      $IF SBC18651
```

292010-54

Traffic Simulator/Monitor Station Program (Continued)

```
            call set$interrupt(INT$7, isr7);
            output (PIC$MASK$130) = ENABLE$586$186;
            output (PIC$EOI$130) = EOI$CMD0$130;
            output (PIC$EOI$130) = EOI$CMD4$130;
            output (PIC$EOI$186) = EOI$CMD0$186;
            output (PIC$VTR$186) = 30H;

$ELSE

            output (PIC$EOI$186) = EOI$CMD0$186;
            output (PIC$EOI$186) = EOI$CMD4$186;
            output (PIC$MASK$186) = ENABLE$586;

$ENDIF

            /* locate iscp */

            iscp$ptr = ISCP$LOC$LO;

            /* set up fields in ISCP */

            iscp.busy = 1;
            iscp.scb$b(0) = SCB$BASE$LO;
            iscp.scb$b(1) = SCB$BASE$HI;
            iscp.scb$o = offset (@scb.status);

            /* set up SCB */

            scb.status = 0;
            scb.cbl$offset = offset (@diagnose.status);
            scb.rpa$offset = offset (@rfd(0).status);
            scb.crc$errs = 0;
            scb.aln$errs = 0;
            scb.rsc$errs = 0;
            scb.ovrn$errs = 0;

            /* set up Diagnose command */

            diagnose.status = 0;
            diagnose.cmd = 7;
            diagnose.link$offset = offset (@configure.status);

            /* set up CONFIGURE command */

            configure.status = 0;
            configure.cmd = 2;
            configure.link$offset = offset (@ia$setup.status);;
            call setup$configure$parameters;


            /* set up IA command */

            ia$setup.status = 0;
            ia$setup.cmd = 1;
            ia$setup.link$offset = offset (@mc$setup.status);
            call setup$ia$parameters;
```

292010-55

Traffic Simulator/Monitor Station Program (Continued)

```
                /* set up MC command */

2               mc$setup.status = 0;
2               mc$setup.cmd = 8003H;
2               mc$setup.link$offset = 0FFFFH;
2               call setup$mc$parameters;


                /* set up one transmit cb linked to itself */

2               transmit.status = 0;
2               call writeln(0, @(0DH, 0AH, ' Would you like to transmit?'), 30, @status);
2               call write(0, @(' Enter a Y or N ==> '), 20, @status);
2               if yes then
2               do;
3                   transmit.cmd = 8004H;
3                   transmit.link$offset = 0FFFFH;
3                   transmit.bd$offset = offset (@tbd.act$count);
3                   call setup$tx$parameters;
3               end;
2               else no$transmission = true;


                /* initialize receive packet area */

2               do i = 0 to 3;
3                   rfd(i).status = 0;
3                   rfd(i).el$s = 0;
3                   rfd(i).link$offset = offset (@rfd(i+1).status);
3                   rfd(i).bd$offset = 0FFFFH;
3                   rbd(i).act$count = 0;
3                   rbd(i).next$bd$link = offset (@rbd(i+1).act$count);
3                   rbd(i).ad0 = offset (@rbuf(i).buffer(0));
3                   rbd(i).ad1 = 0;
3                   rbd(i).size = 1500;
3               end;
2               rfd(0).bd$offset = offset (@rbd(0).act$count);
2               rfd(4).status = 0;
2               rfd(4).el$s = 0;
2               rfd(4).link$offset = offset (@rfd(0).status);
2               rfd(4).bd$offset = 0FFFFH;
2               rbd(4).act$count = 0;
2               rbd(4).next$bd$link = offset (@rbd(0).act$count);
2               rbd(4).ad0 = offset (@rbuf(4).buffer(0));
2               rbd(4).ad1 = 0;
2               rbd(4).size = 1500;


                /* initialize counters */

2               count = 0;
2               receive$count = 0;
2               current$frame = 0;

                /* issue the first CA */
```

292010-56

Traffic Simulator/Monitor Station Program (Continued)

```
724   2          output(CA$PORT) = CA;

725   2      end init;


726   1      print$help: procedure;

727   2          call writeln(0, @(0DH, 0AH, ' Commands are: '), 16, @status);
728   2          call writeln(0, @(0DH, 0AH, ' S - Setup CB                  D - Display RFD/CB'),
                                                                               45, @status);
729   2          call writeln(0, @(' P - Print SCB                C - SCB Control CMD'), 44,
                                                                                   @status);
730   2          call writeln(0, @(' L - ESI Loopback On        N - ESI Loopback Off'), 45,
                                                                                   @status);
731   2          call writeln(0, @(' A - Toggle Number Base'), 23,
                                                                                   @status);
732   2          call writeln(0, @(' Z - Clear Tx Frame Counter'), 27, @status);
733   2          call writeln(0, @(' Y - Clear Rx Frame Counter'), 27, @status);
734   2          call writeln(0, @(' E - Exit to Continuous Mode'), 28, @status);

735   2      end print$help;



736   1      enter$scb$cmd: procedure;
737   2      declare i byte;

             /* enter a command into the SCB */

738   2          call cr$lf;
739   2          if scb.cmd <> 0 then
740   2          do;
741   3              call writeln(0, @(' SCB command word is not cleared'), 32, @status);
742   3              call write(0, @(' Try a Channel Attention? (Y or N) ==> '),
                                                                          39, @status);
743   3              if yes then
744   3              do;
745   4                  output(CA$PORT) = CA;
746   4                  call writeln(0, @(' Issued channel attention'), 25, @status);
747   4                  call cr$lf;
748   4                  return;
749   4              end;
750   3          end;
751   2          call write(0, @(' Do you want to enter any SCB commands? (Y or N) ==> '),
                                                                       53, @status);
752   2          if not yes then return;
754   2          call write(0, @(0DH, 0AH, ' Enter CUC ==> '), 17, @status);
755   2          i = read$int(4);
756   2          scb.cmd = scb.cmd or shl(double(i), 8);
757   2          if i = 1 then scb.cbl$offset = cur$cb$offset;
759   2          call write(0, @(0DH, 0AH, ' Enter RES bit ==> '), 21, @status);
760   2          i = read$bit;
761   2          scb.cmd = scb.cmd or shl(i, 7);
762   2          call write(0, @(0DH, 0AH, ' Enter RUC ==> '), 17, @status);
763   2          i = read$int(4);
764   2          scb.cmd = scb.cmd or shl(i, 4);
```

292010-57

Traffic Simulator/Monitor Station Program (Continued)

```
765   2          if (((scb.cbl$offset = offset (@transmit.status))
                     and ((scb.cmd and 0100H) = 0100H)) or ((scb.cmd and 0010H) = 0010H))
                     and not ((scb.cmd and 0080H) = 0080H)
766   2              then goback = 1;
767   2          call writeln(0, @(0DH, 0AH, ' Issued Channel Attention'), 27, @status);
768   2          call cr$lf;
769   2          output(CA$PORT) = CA;

770   2      end enter$scb$cmd;


771   1      print$type$help: procedure;

772   2          call writeln(0, @(0DH, 0AH, 0AH, 'Command block type: '), 22, @status);
773   2          call writeln(0, @(' N - Nop                    I - IA Setup'), 35, @status);
774   2          call writeln(0, @(' C - Configure              M - MA Setup'), 35, @status);
775   2          call writeln(0, @(' T - Transmit               R - TDR'), 30, @status);
776   2          call writeln(0, @(' D - Diagnose               S - Dump Status'), 38, @status);
777   2          call writeln(0, @(' H - Print this message'), 23, @status);

778   2      end print$type$help;


779   1      setup$cb: procedure;
780   2      declare (t, valid) byte;

781   2          valid = false;
782   2          do while not valid;
783   3              call write(0, @(0DH, 0AH, ' Enter command block type (H for',
                                               ' help) ==> '), 45, @status);
784   3              t = read$char;
785   3              if (t <> 'H') and (t <> 'h') and (t <> 'T') and (t <> 't') and
                         (t <> 'N') and (t <> 'n') and (t <> 'R') and (t <> 'r') and
                         (t <> 'D') and (t <> 'd') and (t <> 'C') and (t <> 'c') and
                         (t <> 'I') and (t <> 'i') and (t <> 'M') and (t <> 'm') and
                         (t <> 'S') and (t <> 's') then
786   3              call write(0, @(0DH, 0AH, ' Illegal command block type'), 29,
                                                                              @status);
787   3              else
788   3              if (t = 'H') or (t = 'h') then call print$type$help;
789   3              else valid = true;
790   3          end;
791   2          if (t = 'N') or (t = 'n') then
792   2          do;
793   3              cur$cb$offset = offset (@nop.status);
794   3              nop.status = 0;
795   3              nop.cmd = 8000H;
796   3              nop.link$offset = 0FFFFH;
797   3          end;
798   2          if (t = 'I') or (t = 'i') then
799   2          do;
800   3              cur$cb$offset = offset (@ia$setup.status);
801   3              ia$setup.status = 0;
802   3              ia$setup.cmd = 8001H;
803   3              ia$setup.link$offset = 0FFFFH;
804   3              call setup$ia$parameters;
805   3          end;
```

292010-58

Traffic Simulator/Monitor Station Program (Continued)

```
806   2          if (t = 'C') or (t = 'c') then
807   2          do;
808   3              cur$cb$offset = offset (@configure.status);
809   3              configure.status = 0;
810   3              configure.cmd = 8002H;
811   3              configure.link$offset = 0FFFFH;
812   3              call setup$configure$parameters;
813   3          end;
814   2          if (t = 'M') or (t = 'm') then
815   2          do;
816   3              cur$cb$offset = offset (@mc$setup.status);
817   3              mc$setup.status = 0;
818   3              mc$setup.cmd = 8003H;
819   3              mc$setup.link$offset = 0FFFFH;
820   3              call setup$mc$parameters;
821   3          end;
822   2          if (t = 'T') or (t = 't') then
823   2          do;
824   3              cur$cb$offset = offset (@transmit.status);
825   3              transmit.status = 0;
826   3              transmit.cmd = 8004H;
827   3              transmit.link$offset = 0FFFFH;
828   3              call setup$tx$parameters;
829   3          end;
830   2          if (t = 'R') or (t = 'r') then
831   2          do;
832   3              cur$cb$offset = offset (@tdr.status);
833   3              tdr.status = 0;
834   3              tdr.cmd = 8005H;
835   3              tdr.link$offset = 0FFFFH;
836   3              tdr.result = 0;
837   3          end;
838   2          if (t = 'S') or (t = 's') then
839   2          do;
840   3              cur$cb$offset = offset (@dump.status);
841   3              dump.status = 0;
842   3              dump.cmd = 8006H;
843   3              dump.link$offset = 0FFFFH;
844   3              dump.buff$ptr = offset (@dump$area(0));
845   3          end;
846   2          if (t = 'D') or (t = 'd') then
847   2          do;
848   3              cur$cb$offset = offset (@diagnose.status);
849   3              diagnose.status = 0;
850   3              diagnose.cmd = 8007H;
851   3              diagnose.link$offset = 0FFFFH;
852   3          end;

853   2      end setup$cb;


854   1      display$command$block: procedure;
855   2      declare (i, j) byte,
                     wh pointer,
                     sel selector,
                     w word;
```

292010-59

Traffic Simulator/Monitor Station Program (Continued)

```
856   2        call cr$lf;
857   2        if cur$cb$offset = OFFFFH then
858   2            call write(0, @(' No Command Block to display'), 28, @status);
859   2        if cur$cb$offset = offset (@nop.status) then
860   2        do;
861   3            call write(0, @('---NOP Command Block---'), 23, @status);
862   3            call print$wds(@nop.status, 3);
863   3        end;
864   2        if cur$cb$offset = offset (@tdr.status) then
865   2        do;
866   3            call write(0, @('---TDR Command Block---'), 23, @status);
867   3            call print$wds(@tdr.status, 4);
868   3        end;
869   2        if cur$cb$offset = offset (@diagnose.status) then
870   2        do;
871   3            call write(0, @('---Diagnose Command Block---'), 28, @status);
872   3            call print$wds(@diagnose.status, 3);
873   3        end;
874   2        if cur$cb$offset = offset (@transmit.status) then
875   2        do;
876   3            call write(0, @('---Transmit Command Block---'), 28, @status);
877   3            if not address$length then i = address$length;
879   3            else i = address$length + 1;
880   3            if ad$loc then call print$wds(@transmit.status, 4);
882   3            else call print$wds(@transmit.status, i/2+1);
883   3            call cr$lf;
884   3            call cr$lf;
885   3            if transmit.bd$offset <> OFFFFH then
886   3            do;
887   4                call write(0, @('---Transmit Buffer Descriptor---'), 33, @status);
888   4                call print$wds(@tbd.act$count, 4);
889   4                call write(0, @(ODH, OAH, OAH,
                           ' Display the transmit buffer? (Y or N) ==> '), 46, @status);
890   4                if yes then
891   4                do;
892   5                    call cr$lf;
893   5                    call writeln(0, @(' Transmit Buffer: '), 17, @status);
894   5                    w = tbd.act$count and 3FFFH;
895   5                    call print$buff(@tx$buffer(0), w);
896   5                end;
897   4            end;
898   3        end;
899   2        if cur$cb$offset = offset (@ia$setup.status) then
900   2        do;
901   3            call write(0, @('---IA Setup Command Block---'), 28, @status);
902   3            call print$wds(@ia$setup.status, 6);
903   3        end;
904   2        if cur$cb$offset = offset (@configure.status) then
905   2        do;
906   3            call write(0, @('---Configure Command Block---'), 29, @status);
907   3            call print$wds(@configure.status, 9);
908   3        end;
909   2        if cur$cb$offset = offset (@mc$setup.status) then
910   2        do;
911   3            call write(0, @('---MC Setup Command Block---'), 28, @status);
912   3            i = 4 + mc$count/2;
```

292010-60

Traffic Simulator/Monitor Station Program (Continued)

```
                if mc$count > 24 then
                    do;
                        call print$wds(@mc$setup.status, 16);
                        call pause;
                        i = i - 16;
                        call print$wds(@mc$setup.mc$address(8), i);
                    end;
                else call print$wds(@mc$setup.status, i);
            end;
            if cur$cb$offset = offset (@dump.status) then
            do;
                call write(0, @('---Dump Status Command Block---'), 31, @status);
                call print$wds(@dump.status, 4);
                if dump.status = 0A000H then
                do;
                    call writeln(0, @(0DH, 0AH, ' Dump Status Results'), 22, @status)
                    call write$offset(@dump$area(0));
                    call cr$lf;
                    do i = 0 to 9;
                        call print$str(@dump$area(16*i), 16);
                    end;
                    call print$str(@dump$area(160), 10);
                    call cr$lf;
                end;
            end;

        end display$command$block;


        display$receive$area: procedure;
        declare (i, k, j, l) byte,
                chars(4) byte;

            call writeln(0, @(0DH, 0AH, ' Frame Descriptors:'), 21, @status);
            if ad$loc then
            do;
                call writeln(0, @(0DH, 0AH, ' DA, SA, and TYPE are in buffer.', 0DH,
                                                                          0AH), 36, @status);
                j = 3;
            end;
            else j = address$length + 4;
            do k = 0 to j;
                do i = 0 to 4;
                    call out$word(@rfd(i).status, k);
                    if k = 0 then call write$offset(@rfd(i).status);
                        else call loop$char(10, ' ');
                end;
                call cr$lf;
            end;
            call writeln(0, @(0DH, 0AH, 0AH, ' Receive Buffer Descriptors:'), 31,
                                                                                @status);
            do k = 0 to 4;
                do i = 0 to 4;
                    call out$word(@rbd(i).act$count, k);
                    if k = 0 then call write$offset(@rbd(i).act$count);
                        else call loop$char(10, ' ');
                end;
```

292010-61

Traffic Simulator/Monitor Station Program (Continued)

```
965   3              call cr$lf;
966   3          end;
967   2          call write(0, @(0DH, 0AH, 0AH, ' Display the receive',
                                               ' buffers? (Y or N) ==> '), 46, @status);
968   2          if not yes then return;
970   2          call writeln(0, @(0DH, 0AH, ' Receive Buffers.'), 19, @status);
971   2          do i = 0 to 4;
972   3              call write(0, @(0DH, 0AH, ' Receive Buffer '), 18, @status);
973   3              call write$int(i, 0);
974   3              call writeln(0, @(' : '), 2, @status);
975   3              k = rbd(i).act$count and 3FFFH;
976   3              call print$buff(@rbuf(i).buffer(0), k);
977   3              call pause;
978   3          end;

979   2      end display$receive$area;


980   1      display$cb$rpa: procedure;
981   2      declare i byte;

982   2          call write(0, @(0DH, 0AH, ' Command Block or Receive Area? (R or C) ==> '),
                                                                         47, @status);
983   2          i = read$char;
984   2          do while (i <> 'R') and (i <> 'r') and (i <> 'C') and (i <> 'c');
985   3              call writeln(0, @(0DH, 0AH, ' Illegal command'), 18, @status);
986   3              call write(0, @(' Enter R or C ==> '), 18, @status);
987   3              i = read$char;
988   3          end;
989   2          if (i = 'R') or (i = 'r') then call display$receive$area;
991   2          else call display$command$block;

992   2      end display$cb$rpa;


993   1      process$cmd: procedure;
994   2      declare (b, i) byte;

995   2          goback = 0;
996   2          b = read$char;
997   2          call cr$lf;
998   2          if (b <> 'H') and (b <> 'h') and (b <> 'S') and (b <> 's') and
                   (b <> 'D') and (b <> 'd') and (b <> 'P') and (b <> 'p') and
                   (b <> 'C') and (b <> 'c') and (b <> 'E') and (b <> 'e') and
                   (b <> 'L') and (b <> 'l') and (b <> 'N') and (b <> 'n') and
                   (b <> 'Z') and (b <> 'z') and (b <> 'Y') and (b <> 'y') and
                   (b <> 'A') and (b <> 'a') then
999   2          call write(0, @(' Illegal command'), 16, @status);
1000  2          if (b = 'H') or (b = 'h') then call print$help;
1002  2          if (b = 'A') or (b = 'a') then
1003  2             if dhex then
1004  2             do;
1005  3                 dhex = false;
1006  3                 call write(0, @(' Counters are displayed in decimal.'), 35,
                                                                          @status);
1007  3             end;
1008  2             else
```

292010–62

Traffic Simulator/Monitor Station Program (Continued)

```
                   do;
1009    3               dhex = true;
1010    3               call write(0, @(' Counters are displayed in hexadecimal. '), 39,
                                                                        @status);
1011    3           end;
1012    2       if (b = 'L') or (b = 'l') then
1013    2       do;
1014    3           output(ESI$PORT) = LOOPBACK;
1015    3           call write(0, @(' ESI is in Loopback Mode. '), 25, @status);
1016    3       end;
1017    2       if (b = 'N') or (b = 'n') then
1018    2       do;
1019    3           output(ESI$PORT) = NO$LOOPBACK;
1020    3           call write(0, @(' ESI is NOT in Loopback Mode. '), 29, @status);
1021    3       end;
1022    2       if (b = 'Z') or (b = 'z') then
1023    2       do;
1024    3           count = 0;
1025    3           call write(0, @(' Transmit Frame Counter is cleared. '), 35, @status);
1026    3       end;
1027    2       if (b = 'Y') or (b = 'y') then
1028    2       do;
1029    3           receive$count = 0;
1030    3           scb.crc$errs, scb.aln$errs, scb.rsc$errs, scb.ovrn$errs = 0;
1031    3           call write(0, @(' Receive Frame Counter is cleared. '), 34, @status);
1032    3       end;
1033    2       if (b = 'C') or (b = 'c') then call enter$scb$cmd;
1035    2       if (b = 'S') or (b = 's') then call setup$cb;
1037    2       if (b = 'P') or (b = 'p') then call print$scb;
1039    2       if (b = 'D') or (b = 'd') then call display$cb$rpa;
1041    2       if (b = 'E') or (b = 'e') then goback = 1;
1043    2       call cr$lf;

1044    2   end process$cmd;


1045    1   getout: procedure;
1046    2   declare b byte;

1047    2       b = read$char;
1048    2       goback = 0;
1049    2       call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                        @status);
1050    2       do forever;
1051    3           if csts then
1052    3           do;
1053    4               disable;
1054    4               call process$cmd;
1055    4               enable;
1056    4               if goback then return;
1058    4               call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                        @status);
1059    4           end;
1060    3       end;

1061    2   end getout;
```

292010-63

Traffic Simulator/Monitor Station Program (Continued)

```
1062   1        update: procedure;
1063   2        declare i byte;

1064   2            call cr$lf;
1065   2            call loop$char(10, 0AH);
1066   2            call loop$char(28, '*');
1067   2            call write(0, @(' Station Configuration '), 23, @status);
1068   2            call loop$char(27, '*');
1069   2            call cr$lf;
1070   2            call cr$lf;
1071   2            call write(0, @(' Host Address:  '), 15, @status);
1072   2            call print$network$addr(@ia$setup.ia$address(0));
1073   2            i = 0;
1074   2            call write(0, @(' Multicast Address(es):  '), 24, @status);
1075   2            if mc$setup.mc$byte$count = 0
1076   2            then call writeln(0, @('No Multicast Addresses Defined'), 30, @status);
1077   2            else
                        do while i < mc$setup.mc$byte$count;
1078   3                    call print$network$addr(@mc$setup.mc$address(i));
1079   3                    call loop$char(24, ' ');
1080   3                    i = i + 6;
1081   3                end;
1082   2            call write(0, @(0DH), 1, @status);
1083   2            if not no$transmission then call print$parameters;
1085   2            call write(0, @(' 82586 Configuration Block:  '), 28, @status);
1086   2            call print$str(@configure.info(0), 10);
1087   2            call cr$lf;
1088   2            call loop$char(29, '*');
1089   2            call write(0, @(' Station Activities '), 20, @status);
1090   2            call loop$char(29, '*');
1091   2            call cr$lf;
1092   2            call cr$lf;
1093   2            call writeln(0,
                @(' # of Good     # of Good     CRC           Alignment     No            Receive'),
                                                                                      73, @status);
1094   2            call writeln(0,
                @(' Frames        Frames        Errors        Errors        Resource      Overrun'),
                                                                                      73, @status);
1095   2            call writeln(0,
                @(' Transmitted   Received                                  Errors        Errors'),
                                                                                      72, @status);

1096   2        end update;


1097   1        main:

                    call init;
1098   1            enable;
1099   1            do while reset;
1100   2            end;
1101   1            disable;
1102   1            scb.cmd = 0100H;
1103   1            output(CA$PORT) = CA;
1104   1            call wait$scb;
1105   1            enable;
```

292010-64

Traffic Simulator/Monitor Station Program (Continued)

```
1            do while (diagnose.status and 8000H) <> 8000H;
2            end;
1            call cr$lf;
1            if diagnose.status <> 0A000H
1            then call writeln(0, @(' Diagnose failed!'), 17 , @status);
1            if configure.status <> 0A000H
1            then call writeln(0, @(' Configure failed!'), 18 , @status);
1            if ia$setup.status <> 0A000H
1            then call writeln(0, @(' IA Setup failed!'), 17 , @status);
1            if mc$setup.status <> 0A000H
1            then call writeln(0, @(' MC Setup failed!'), 17 , @status);
1            scb.cbl$offset = offset (@transmit.status);
1            call writeln(0, @(0DH, 0AH, ' Receive Unit is active.'), 26, @status);
1            disable;
1            scb.cmd = 0010H;
1            output(CA$PORT) = CA;
1            call wait$scb;
1            enable;
1            output(ESI$PORT) = NO$LOOPBACK;
1            call cr$lf;
1            if not no$transmission then
1            do;
2                call write(0, @('---Transmit Command Block---'), 28, @status);
2                call print$wds(@transmit.status, 8);
2                call cr$lf;
2                cur$cb$offset = offset (@transmit.status);
2                call pause;
2                do z = 1 to 60;
3                    call time(250);
3                end;
2                call writeln(0, @(0DH, 0AH, 'transmission started!'), 23, @status);
2                call cr$lf;
2                disable;
2                scb.cmd = 0100H;
2                output (CA$PORT) = CA;
2                call wait$scb;
2                enable;
2            end;
1            call update;
1            do forever;
2                call write(0, @(0DH, ' '), 2, @status);
2                do y = 0 to 5;
3                    do case y;
4                        call write$int(count, dhex);
4                        call write$int(receive$count, dhex);
4                        call write$int(scb.crc$errs, dhex);
4                        call write$int(scb.aln$errs, dhex);
4                        call write$int(scb.rsc$errs, dhex);
4                        call write$int(scb.ovrn$errs, dhex);
4                    end;
3                    char$count = 13 - char$count;
3                    call loop$char(char$count, ' ');
3                end;
2                if csts then
2                do;
3                    disable;
3                    call getout;
3                    call update;
3                end;
2            end;

1        end tsms;
```

292010-65

```
MODULE INFORMATION:

     CODE AREA SIZE     = 23C3H   9155D
     CONSTANT AREA SIZE = 0F85H   3973D
     VARIABLE AREA SIZE = 265EH   9822D
     MAXIMUM STACK SIZE = 0092H    146D
     1994 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     159KB MEMORY AVAILABLE
     23KB MEMORY USED   (14%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

292010-66

Traffic Simulator/Monitor Station Program (Continued)

```
          /*************************************************************************
          /*                                                                     */
          /*          186/586 High Integration Board initialization Routine      */
          /*          (This driver is configured for Ethernet/Cheapernet Design  */
          /*           Kit Demo Board)                                           */
          /*                                                                     */
          /*          Ver. 2.0                          March 14, 1986           */
          /*                                                                     */
          /*          Kiyoshi Nishide                   Intel Corporation        */
          /*                                                                     */
          /*************************************************************************/

          /* The conditional compilation parameter 'EPROM27128' determines board ROM
             size.  If it is true, the 80186's wait state generator is programmed to
             0 wait state for upper 64K-byte memory locations.  If it is false, the
             wait state generator is programmed to 0 wait state for upper 128K-byte
             memory locations    */


1         ini186:
          do;

2    1    declare hib_ir label public;
3    1    declare main label external;
4    1    declare menu label external;

          /* literals */

5    1    declare lit          literally 'literally',
                  UMCS_reg           lit 'OFFAOH',
                  LMCS_reg           lit 'OFFA2H',
                  PACS_reg           lit 'OFFA4H',
                  MPCS_reg           lit 'OFFA8H',
                  INT_MASK_reg       lit 'OFF28H',
                  ISCP$LOC$LO        lit '03FF8H',
                  ISCP$LOC$HI        lit '0',
                  SCC_CH_B_CMD       lit '8300H',
                  SCC_CH_B_DATA      lit '8302H',
                  SCC_CH_A_CMD       lit '8304H',
                  SCC_CH_A_DATA      lit '8306H',
                  NUL                lit '0',
                  CR                 lit '0DH',
                  LF                 lit '0AH',
                  BS                 lit '08H',
                  SP                 lit '20H',
                  QM                 lit '3FH',
                  DEL                lit '07FH',
                  BEL                lit '07H';
```

292010-67

186/586 High Integration Board Initialization Routine

```
                 /* System Configuration Pointer */

  6    1         declare scp structure
                         (
                         sysbus byte,
                         unused (5) byte,
                         iscp$addr$lo word,
                         iscp$addr$hi word
                         )
                         at (0FFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);

  7    1         init$int$clt: procedure;

  8    2                 output(INT_mask_reg) = 0FFH; /* mask all interrupts */

  9    2         end init$int$clt;


 10    1         rra: procedure (reg_no) byte;
 11    2         declare reg_no byte;

 12    2             if (reg_no and 0FH) <> 0 then output(SCC_CH_A_CMD) = reg_no and 0FH;
 14    2             return input(SCC_CH_A_CMD);

 15    2         end rra;


 16    1         rrb: procedure (reg_no) byte;
 17    2         declare reg_no byte;

 18    2             if (reg_no and 0FH) <>0 then output (SCC_CH_B_CMD) = reg_no and 0FH;
 20    2             return input(SCC_CH_B_CMD);

 21    2         end rrb;


 22    1         wra: procedure (reg_no, value);
 23    2         declare (reg_no, value) byte;

 24    2             if (reg_no and 0FH) <> 0 then output (SCC_CH_A_CMD) = reg_no and 0FH;
 26    2             output (SCC_CH_A_CMD) = value;

 27    2         end wra;

 28    1         wrb: procedure (reg_no, value);
 29    2         declare (reg_no, value) byte;

 30    2             if (reg_no and 0FH) <> 0 then output (SCC_CH_B_CMD) = reg_no and 0FH;
 32    2             output (SCC_CH_B_CMD) = value;

 33    2         end wrb;

 34    1         init$SCC$B: procedure;

 35    2             call wrb(09, 01000000b);   /* channel B reset */
```

292010-68

186/586 High Integration Board Initialization Routine (Continued)

```
36  2         call wrb(04, 01001110b);    /* 2 stop, no parity, brf = 16x */
37  2         call wrb(03, 11000000b);    /* rx 8 bits/char, no auto-enable */
38  2         call wrb(05, 01100000b);    /* tx 8 bits/char */
39  2         call wrb(10, 00000000b);
40  2         call wrb(11, 01010110b);    /* rxc = txc = BRG, trxc = BRG out */
41  2         call wrb(12, 00001011b);    /* baud rate = 9600 */
42  2         call wrb(13, 00000000b);
43  2         call wrb(14, 00000011b);    /* BRG source = SYS CLK, enable BRG */
44  2         call wrb(15, 00000000b);    /* all ext status interrupts off */

45  2         call wrb(03, 11000001b);    /* scc-b receive enable */
46  2         call wrb(05, 11101010b);    /* scc-b transmit enable, dtr on, rts on */

47  2     end init$SCC$B;


48  1     c$in: procedure byte public;

49  3         do while (input(SCC_CH_B_CMD) and 1) = 0; end;
51  2         return (input(SCC_CH_B_DATA));

52  2     end c$in;


53  1     c$out: procedure (char) public;
54  2     declare char byte;

55  3         do while (input(SCC_CH_B_CMD) and 4) = 0; end;
57  2         output(SCC_CH_B_DATA) = char;

58  2     end c$out;


59  1     read: procedure (file$id, msg$ptr, count, actual$ptr, status$ptr) public;
60  2     declare file$id word,
                  msg$ptr pointer,
                  count word,
                  actual$ptr pointer,
                  status$ptr pointer,
                  msg based msg$ptr (1) byte,
                  buf (200) byte,
                  actual based actual$ptr word,
                  status based status$ptr word,
                  i word,
                  ch byte;

      /* This procedure implements the ISIS read procedure. All control characters */
      /* except LF, BS, and DEL are ignored. If BS or DEL is encountered, a         */
      /* backspace is done.                                                         */

61  2         status = 0;
62  2         i, ch = 0;
63  2         do while (ch <> CR) and (ch <> LF) and (i < 198);
64  3             ch = c$in and 07FH;
65  3             if (ch = BS) or (ch = DEL) then
66  3             do;
```

292010–69

186/586 High Integration Board Initialization Routine (Continued)

```
67   4                    if i > 0 then
68   4                    do;
69   5                        i = i - 1;
70   5                        call c$out(DEL);
71   5                        call c$out(BS);
72   5                        call c$out(SP);
73   5                        call c$out(DEL);
74   5                        call c$out(BS);
75   5                    end;
76   4                    else
                              call c$out(BEL);
77   4                end;
78   3                else
                          if ch >= SP then
79   3                    do;
80   4                        call c$out(ch);
81   4                        buf(i) = ch;
82   4                        i = i + 1;
83   4                    end;
84   3                else
                          if (ch = CR) or (ch = LF) then
85   3                    do;
86   4                        buf(i) = CR;
87   4                        buf(i + 1) = LF;
88   4                        i = i + 2;
89   4                    end;
90   3                else
                          call c$out(BEL);
91   3            end;
92   2            call c$out(CR);
93   2            if i > count then i = count;
95   2            actual = i;
96   2            do i = 0 to actual - 1;
97   3                msg(i) = buf(i);
98   3            end;

99   2        end read;


100  1        csts: procedure byte public;

101  2            return ((input(SCC_CH_B_CMD) and 1) <> 0);

102  2        end csts;


103  1        write: procedure (file$id, msg$ptr, count, status$ptr) public;
104  2        declare (file$id, count) word,
                      (msg$ptr, status$ptr) pointer,
                      msg based msg$ptr (1) byte,
                      status based status$ptr word,
                      ch byte,
                      i word;

              /* This procedure implements the ISIS write */

105  2            status = 0;
```

292010-70

186/586 High Integration Board Initialization Routine (Continued)

```
106   2        i = 0;
107   2        do while i < count;
108   3            ch = msg(i);
109   3            if ((ch >= SP) and (ch < DEL)) or (ch = CR) or (ch = LF) or (ch = NUL)
                   then
110   3                call c$out(ch);
111   3            else
                       call c$out(QM);
112   3            i = i + 1;
113   3        end;

114   2    end write;


115   1    hib_ir:

           $IF EPROM27128

                   output(UMCS_reg) = 0F038H;    /* Starting Address = 0F0000H,
                                                    no wait state */

           $ELSE

                   output(UMCS_reg) = 0E038H;    /* Starting Address = 0E0000H,
                                                    no wait state */

           $ENDIF

116   1            output(LMCS_reg) = 03FCH;     /* 16K, no wait state */
117   1            output(PACS_reg) = 083CH;     /* PBA = 8000H, no wait state for
                                                    PSC0-3 */
118   1            output(MPCS_reg) = 0BFH;      /* Peripherals in I/O space, no A1 & A2
                                                    provided, 3 wait states for PSC4-6 */

119   1            call init$int$clt;
120   1            call init$SCC$B;
121   1            go to main;

122   1    end ini186;
```

292010-71

186/586 High Integration Board Initialization Routine (Continued)

APPENDIX C
THE 82530 SCC - 80186 INTERFACE AP BRIEF

INTRODUCTION

The object of this document is to give the 82530 system designer an in-depth worst case design analysis of the typical interface to a 80186 based system. This document has been revised to include the new specifications for the 6 MHz 82530. The new specifications yield better margins and a 1 wait state interface to the CPU (2 wait states are required for DMA cycles). These new specifications will appear in the 1987 data sheet and advanced specification information can be obtained from your local Intel sales office. The following analysis includes a discussion of how the interface TTL is utilized to meet the timing requirements of the 80186 and the 82530. In addition, several optional interface configurations are also considered.

INTERFACE OVERVIEW

The 82530 - 80186 interface requires the TTL circuitry illustrated in Figure 1. Using five 14 pin TTL packages, 74LS74, 74AS74, 74AS08, 74AS04, and 74LS32, the following operational modes are supported:

- Polled
- Interrupt in vectored mode
- Interrupt in non-vectored mode
- Half-duplex DMA on both channels
- Full-duplex DMA on channel A

A brief description of the interface functional requirements during the five possible BUS operations follows below.

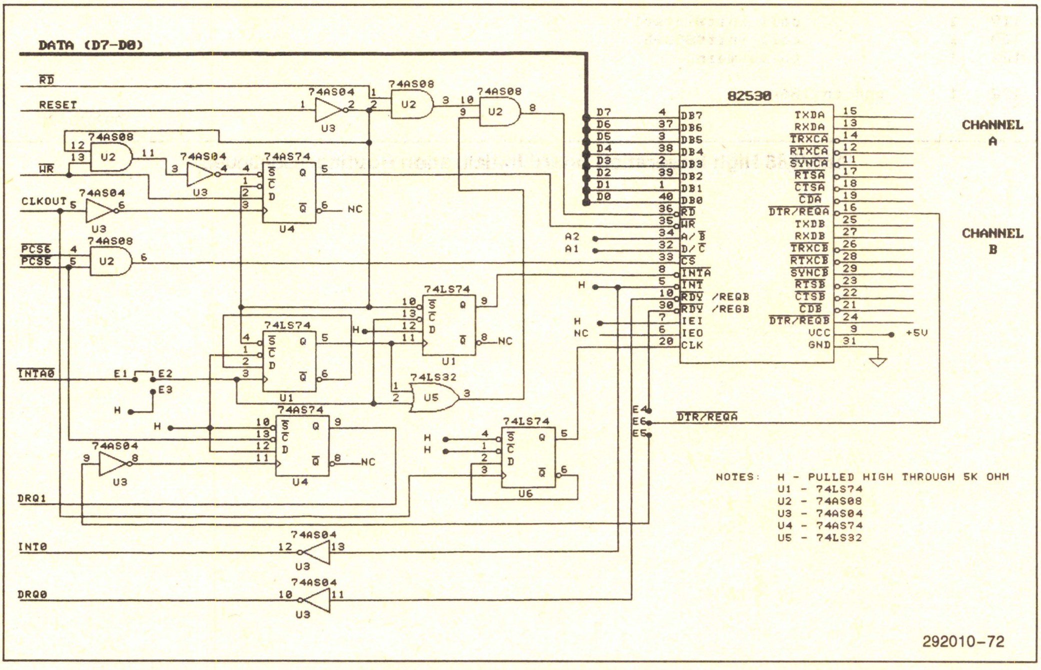


Figure 1. 82530-80186 Interface

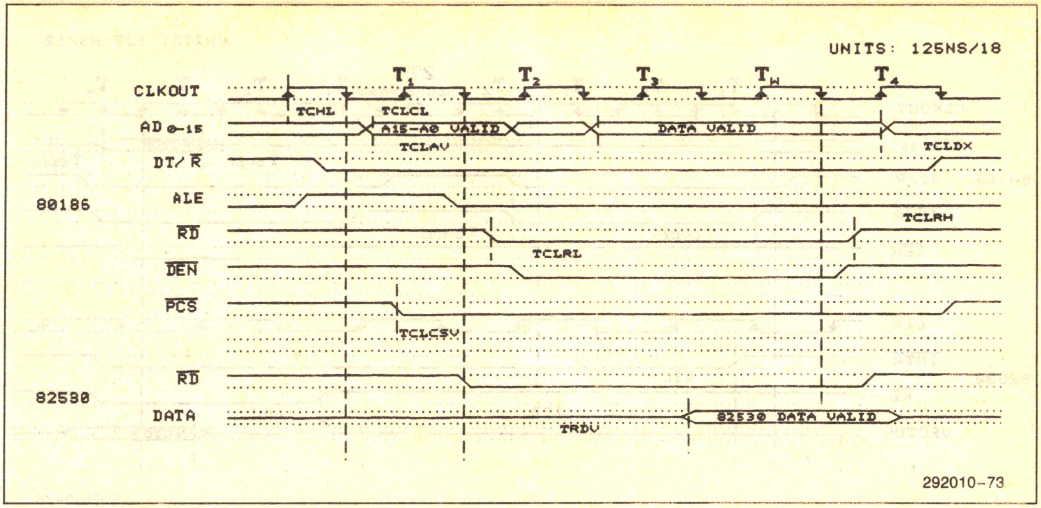


Figure 2. 80186-82530 Interface Read Cycle

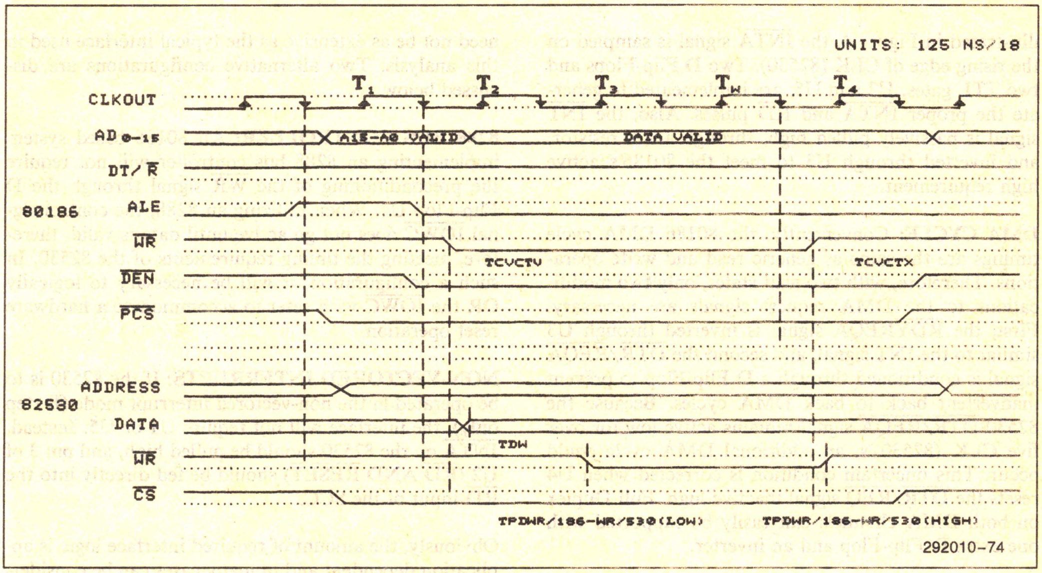


Figure 3. 80186-82530 Interface Write Cycle

READ CYCLE: The 80186 read cycle requirements are met without any additional logic, Figure 2. At least one wait state is required to meet the 82530 tAD access time.

WRITE CYCLE: The 82530 requires that data must be valid while the WR pulse is low, Figure 3. A D Flip-Flop delays the leading edge of WR until the falling edge of CLOCKOUT when data is guaranteed valid and WR is guaranteed active. The CLOCKOUT signal is inverted to assure that WR is active low before the D Flip-Flop is clocked. No wait states are necessary to meet the 82530's WR cycle requirements, but one is assumed from the RD cycle.

INTA CYCLE: During an interrupt acknowledge cycle, the 80186 provides two INTA pulses, one per bus cycle, separated by two idle states. The 82530 expects only one long INTA pulse with a RD pulse occurring only after the 82530 IEI/IEO daisy chain settles. As

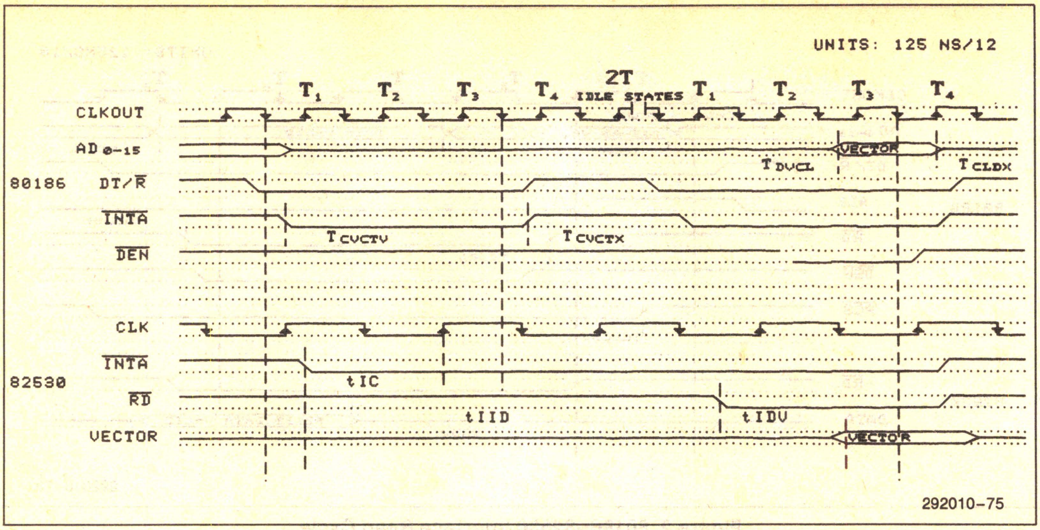


Figure 4. 82530–80186 $\overline{INTA}$ Cycle

illustrated in Figure 4, the $\overline{INTA}$ signal is sampled on the rising edge of CLK (82530). Two D Flip-Flops and two TTL gates, U2 and U5, are implemented to generate the proper $\overline{INTA}$ and $\overline{RD}$ pulses. Also, the $\overline{INT}$ signal is passively pulled high, through a 1 k resistor, and inverted through U3 to meet the 80186's active high requirement.

DMA CYCLE: Conveniently, the 80186 DMA cycle timings are the same as generic read and write operations. Therefore, with two wait states, only two modifications to the DMA request signals are necessary. First, the $\overline{RDY}$REQA signal is inverted through U3 similar to the INT signal, and second the $\overline{DTR/REQA}$ signal is conditioned through a D Flip-Flop to prevent inadvertent back to back DMA cycles. Because the 82530 $\overline{DTR/REQA}$ signal remains active low for over five CLK (82530)'s, an additional DMA cycle could occur. This uncertain condition is corrected when U4 resets the $\overline{DTR/REQ}$ signal inactive high. Full Duplex on both DMA channels can easily be supported with one extra D Flip-Flop and an inverter.

RESET: The 82530 does not have a dedicated RESET input. Instead, the simultaneous assertion of both $\overline{RD}$ and $\overline{WR}$ causes a hardware reset. This hardware reset is implemented through U2, U3, and U4.

ALTERNATIVE INTERFACE CONFIGURATIONS

Due to its wide range of applications, the 82530 interface can have many varying configurations. In most of these applications the supported modes of operation need not be as extensive as the typical interface used in this analysis. Two alternative configurations are discussed below.

8288 BUS CONTROLLER: An 80186 based system implementing an 8288 bus controller will not require the preconditioning of the $\overline{WR}$ signal through the D Flip-Flop U4. When utilizing an 8288, the control signal $\overline{IOWC}$ does not go active until data is valid, therefore, meeting the timing requirements of the 82530. In such a configuration, it will be necessary to logically OR the $\overline{IOWC}$ with reset to accommodate a hardware reset operation.

NON-VECTORED INTERRUPTS: If the 82530 is to be operated in the non-vectored interrupt mode (B step only), the interface will not require U1 or U5. Instead, $\overline{INTA}$ on the 82530 should be pulled high, and pin 3 of U2 ($\overline{RD}$ AND $\overline{RESET}$) should be fed directly into the $\overline{RD}$ input of the SCC.

Obviously, the amount of required interface logic is application dependent and in many cases can be considerably less than required by the typical configuration, supporting all modes of SCC operation.

DESIGN ANALYSIS

This design analysis is for a typical microprocessor system, pictured in Figure 5. The Timing analysis assumes an 8 MHz 80186 and a 4 MHz 82530. Also, included in the analysis are bus loading, and TTL-MOS compatibility considerations.

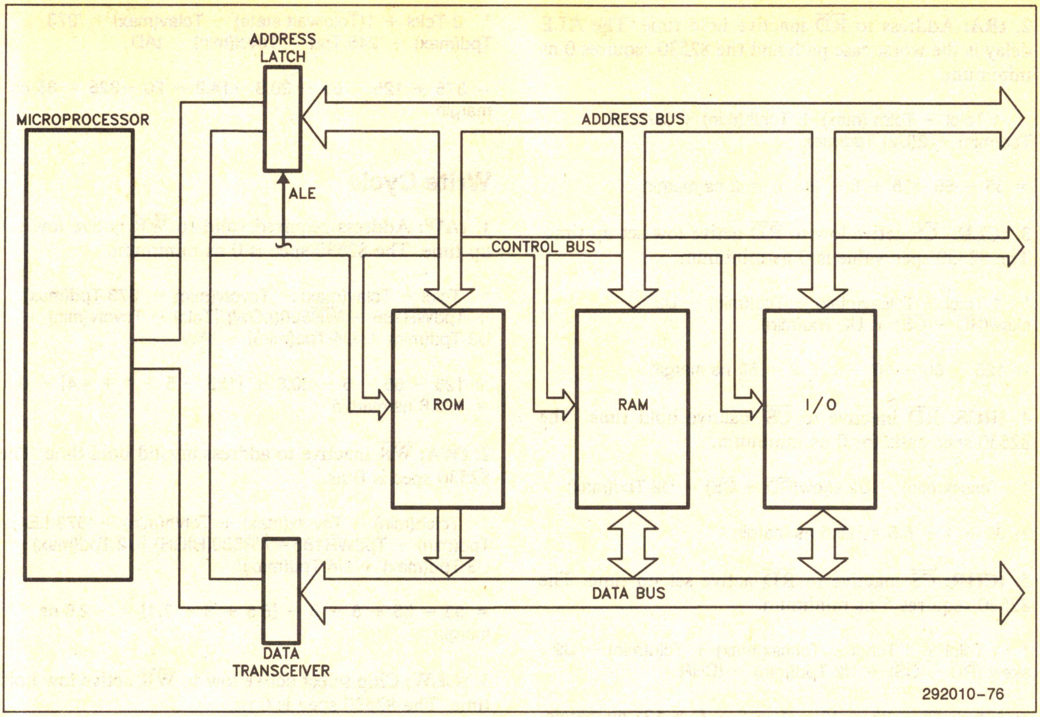


Figure 5. Typical Microprocessor System

Bus Loading and Voltage Level Compatabilities

The data and address lines do not exceed the drive capability of either 80186 or the 82530. There are several control lines that drive more than one TTL equivalent input. The drive capability of these lines are detailed below.

$\overline{\text{WR}}$: The $\overline{\text{WR}}$ signal drives U3 and U4.

* Iol (2.0 mA) > Iil (−0.4 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{\text{PCS5}}$: The $\overline{\text{PCS5}}$ signal drives U2 and U4.

* Iol (2.0 mA) > Iil (−0.5 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{\text{INTA}}$: The $\overline{\text{INTA}}$ signal drives 2(U1) and U5.

* Iol (2.0 mA) > Iil (−0.4 mA + −0.8 mA + −0.4 mA)
Ioh (−400 μA) > Iih (20 μA + 40 μA + 20 μA)

All the 82530 I/O pins are TTL voltage level compatible.

TIMING ANALYSIS

Certain symbolic conventions are adhered to throughout the analysis below and are introduced for clarity.

1. All timing variables with a lower case first letter are 82530 timing requirements or responses (i.e., tRR).
2. All timing variables with Upper case first letters are 80186 timing responses or requirements unless preceded by another device's alpha-numeric code (i.e., Tclcl or '373 Tpd).
3. In the write cycle analysis, the timing variable Tpd$\overline{\text{WR}}$186-$\overline{\text{WR}}$530 represents the propagation delay between the leading or trailing edge of the $\overline{\text{WR}}$ signal leaving the 80186 and the $\overline{\text{WR}}$ edge arrival at the 82530 $\overline{\text{WR}}$ input.

Read Cycle

1. **tAR:** Address valid to $\overline{\text{RD}}$ active set up time for the 82530. Since the propagation delay is the worst case path in the assumed typical system, the margin is calculated only for a propagation delay constrained and not an ALE limited path. The spec value is 0 ns minimum.

* 1 Tclcl − Tclav(max) − '245 Tpd(max) + Tclrl(min) + 2(U2) Tpd(min) − tAR(min)

= 125 − 55 − 20.8 + 10 + 2(2) − 0 = 63.2 ns margin

2. **tRA:** Address to $\overline{RD}$ inactive hold time. The ALE delay is the worst case path and the 82530 requires 0 ns minimum.

* 1 Tclcl − Tclrh (max) + Tchlh(min) + '373 LE Tpd(min) − 2(U2) Tpd(max)

= 55 − 55 + 5 + 8 − 2(5.5) = 2 ns margin

3. **tCLR:** $\overline{CS}$ active low to $\overline{RD}$ active low set up time. The 82530 spec value is 0 ns minimum.

* 1 Tclcl − Tclcsv(max) − Tclrl(min) − U2 skew($\overline{RD}$ − $\overline{CS}$) + U2 Tpd(min)

= 125 − 66 − 10 − 1 + 2 = 50 ns margin

4. **tRCS:** $\overline{RD}$ inactive to $\overline{CS}$ inactive hold time. The 82530 spec calls for 0 ns minimum.

* Tcscsx(min) − U2 skew($\overline{RD}$ − $\overline{CS}$) − U2 Tpd(max)

= 35 − 1 − 5.5 = 28.5 ns margin

5. **tCHR:** $\overline{CS}$ inactive to $\overline{RD}$ active set up time. The 82530 requires 5 ns minimum.

* 1 Tclcl + 1 Tchcl − Tchcsx(max) + Tclrl(min) − U2 skew ($\overline{RD}$ − $\overline{CS}$) + U2 Tpd(min) − tCHR

= 125 + 55 − 35 − 10 − 1 + 2 − 5 = 131 ns margin

6. **tRR:** $\overline{RD}$ pulse active low time. One 80186 wait state is included to meet the 150 ns minimum timing requirements of the 82530.

* Trlrh(min) + 1(Tclclwait state) − 2(U2 skew) − tRR

= (250−50) + 1(125) − 2(1) − 150 = 173 ns margin

7. tRDV: $\overline{RD}$ active low to data valid maximum delay for 80186 read data set up time (Tdvcl = 20 ns). The margin is calculated on the Propagation delay path (worst case).

* 2 Tclcl + 1(Tclclwait state) − Tclrl(max) − Tdvcl(min) − '245 Tpd(max) − 82530 tRDV(max) − 2(U2) Tpd(max)

= 2(125) + 1(125) − 70 − 20 − 14.2 − 105 − 2(5.5)
= 154 ns margin

8. **tDF:** $\overline{RD}$ inactive to data output float delay. The margin is calculated to $\overline{DEN}$ active low of next cycle.

* 2 Tclcl + Tclch(min) − Tclrh(max) + Tchctv(min) − 2(U2) Tpd(max) − 82530 tDF(max)

= 250 + 55 −55 + 10 − 11 − 70 = 179 ns margin

9. **tAD:** Address required valid to read data valid maximum delay. The 82530 spec value is 325 ns maximum.

* 3 Tclcl + 1(Tclclwait state) − Tclav(max) − '373 Tpd(max) − '245 Tpd − Tdvcl(min) − tAD

= 375 + 125 − 55 − 20.8 −14.2 − 20 −325 = 65 ns margin

Write Cycle

1. **tAW:** Address required valid to $\overline{WR}$ active low set up time. The 82530 spec is 0 ns minimum.

* Tclcl − Tclav(max) − Tcvctv(min) − '373 Tpd(max) + Tpd$\overline{WR}$186 − $\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tAW

= 125 − 55 − 5 − 20.8 + [125 − 5 + 1 + 4.4] − 0
= 170.6 ns margin

2. **tWA:** $\overline{WR}$ inactive to address invalid hold time. The 82530 spec is 0 ns.

* Tclch(min) − Tcvctx(max) + Tchlh(min) + '373 LE Tpd(min) − Tpd$\overline{WR}$186=$\overline{WR}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 55 + 5 + 8 − [5.5 + 3 + 7.1] = −2.6 ns margin

3. **tCLW:** Chip select active low to $\overline{WR}$ active low hold time. The 82530 spec is 0 ns.

* 1 Tclcl − Tclcsv(max) + Tcvctv(min) − U2 Tpd(max) + Tpd$\overline{WR}$186=$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 125 − 66 + 5 − 5.5 + [125 − 5 + 1 + 4.4] =
183.9 ns margin

4. **tWCS:** $\overline{WR}$ invalid to Chip Select invalid hold time. 82530 spec is 0 ns.

* Tcxcsx(min) − U2 Tpd(max) − Tpd$\overline{WR}$186=$\overline{WR}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 35 + 1.5 − [5.5 + 3 + 7.1] = 20.9 ns margin

5. **tCHW:** Chip Select inactive high to $\overline{WR}$ active low set up time. The 82530 spec is 5 ns.

* 1 Tclcl + Tchcl(min) + Tcvctv(min) − Tchcsx(max) − U2 Tpd(max) + Tpd$\overline{WR}$186=$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tCHW

= 125 + 55 + 5 − 35 − 5.5 + [125 −5 + 1 + 4.4] − 5 = 264 ns margin

6. **tWW:** $\overline{WR}$ active low pulse. 82530 requires a minimum of 60 ns from the falling to the rising edge of $\overline{WR}$. This includes one wait state.

* Twlwh [2Tclcl − 40] + 1 (Tclclwait state) − Tpd$\overline{WR}$/186−$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] + Tpd$\overline{WR}$/186=$\overline{WR}$/530(HIGH) [U2 Tpd(min) U3 Tpd(min) + U4 Tpd(min)] − tWW

= 210 + 1(125) − [125 − 5 + 4.5 + 9.2] − [1.5 + 1 + 3.2] − 60 = 135.6 ns margin

7. **tDW:** Data valid to $\overline{WR}$ active low setup time. The 82530 spec requires 0 ns.

* Tcvctv(min) − Tcldv(max) − '245 Tpd(max) + Tpd$\overline{WR}$186−$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 5 − 44 − 14.2 + 125 − 5 + 1.0 + 4.4 = 72.2 ns margin

8. **tWD:** Data valid to $\overline{WR}$ inactive high hold time. The 82530 requires a hold time of 0 ns.

* Tclch − skew {Tcvctx(max) + Tcvctx(min)} + '245 OE Tpd(min) − Tpd$\overline{WR}$186−$\overline{WR}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 5 + 11.25 − [5.5 + 3.0 + 7.1] = −50.6 ns margin

INTA Cycle:

1. **tIC:** This 82530 spec implies that the $\overline{INTA}$ signal is latched internally on the rising edge of CLK (82530). Therefore the maximum delay between the 80186 asserting $\overline{INTA}$ active low or inactive high and the 82530 internally recognizing the new state of $\overline{INTA}$ is the propagation delay through U1 plus the 82530 CLK period.

* U1 Tpd(max) + 82530 CLK period

= 45 + 250 = 295 ns

2. **tCI:** rising edge of CLK to $\overline{INTA}$ hold time. This spec requires that the state of $\overline{INTA}$ remains constant for 100 ns after the rising edge of CLK. If this spec is violated any change in the state of $\overline{INTA}$ may not be internally latched in the 82530. tCI becomes critical at the end of an $\overline{INTA}$ cycle when $\overline{INTA}$ goes inactive. When calculating margins with tCI, an extra 82530 CLK period must be added to the $\overline{INTA}$ inactive delay.

3. **tIW:** $\overline{INTA}$ inactive high to $\overline{WR}$ active low minimum setup time. The spec pertains only to 82530 $\overline{WR}$ cycle and has a value of 55 ns. The margin is calculated assuming an 82530 $\overline{WR}$ cycle occurs immediately after an $\overline{INTA}$ cycle. Since the CPU cycles following an 82530 $\overline{INTA}$ cycle are devoted to locating and executing the proper interrupt service routine, this condition should never exist. 82530 drivers should insure that at least one CPU cycle separates $\overline{INTA}$ and $\overline{WR}$ or $\overline{RD}$ cycles.

4. **tWI:** $\overline{WR}$ inactive high to $\overline{INTA}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{INTA}$.

* Tclcl − Tcvctx(max) − Tpd$\overline{WR}$186 − $\overline{WR}$530(HIGH) [U3 Tpd(max) + U4 Tpd(max)] + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − [5.5 + 3 + 7.1] + 5 + 10 = 69.4 ns margin

5. **tIR:** $\overline{INTA}$ inactive high to $\overline{RD}$ active low minimum setup time. This spec pertains only to 82530 $\overline{RD}$ cycles and has a value of 55 ns. The margin is calculated in the same manner as tIW.

6. **tRI:** $\overline{RD}$ inactive high to $\overline{INTA}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{INTA}$.

* Tclcl − Tclrh(max) − 2 U2 Tpd(max) + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − 2(5.5) + 5 + 10 = 74 ns margin

7. **tIID:** $\overline{INTA}$ active low to $\overline{RD}$ active low minimum setup time. This parameter is system dependent. For any SCC in the daisy chain, tIID must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for this particular SCC, and tEIEO for each device separating them in the daisy chain. The typical system with only 1 SCC requires tIID to be greater than tCEQ. Since tEI occurs coincidently with tCEQ and it is smaller it can be neglected. Additionally, tEIEO does not have any relevance to a system with only one SCC. Therefore tIID > tCEQ = 250 ns.

* 4 Tclcl + 2 Tidle states − Tcvctv(max) − tIC [U1 Tpd(max) + 82530 CLK period] + Tcvctv(min) + U5 Tpd(min) + U2 Tpd(min) − tIID

= 500 + 250 − 70 − [45 + 250] + 5 + 6 + 2 − 250 = 148 ns margin

8. **tIDV:** $\overline{RD}$ active low to interrupt vector valid delay. The 80186 expects the interrupt vector to be valid on the data bus a minimum of 20 ns before T4 of the second acknowledge cycle (Tdvcl). tIDV spec is 100 ns maximum.

* 3 Tclcl − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) − tIDV(max) − '245 Tpd(max) − Tdvcl(min)

= 375 − 70 − 25 − 5.5 − 100 − 14.2 − 20 = 140.3 ns margin

9. **tII:** $\overline{RD}$ pulse low time. The 82530 requires a minimum of 125 ns.

* 3 Tclcl − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) + Tcvctx(min) + U5 Tpd(min) + U2 Tpd(min) − tII(min)

= 375 − 70 − 25 − 5.5 + 5 + 6 + 1.5 − 125 = 162 ns margin

DMA Cycle

Fortunately, the 80186 DMA controller emulates CPU read and write cycle operation during DMA transfers. The DMA transfer timings are satisfied using the above analysis. Because of the 80186 DMA request input requirements, two wait states are necessary to prevent inadvertent DMA cycles. There are also $\overline{CPU}$DMA intracycle timing considerations that need to be addressed.

1. **tDRD:** $\overline{RD}$ inactive high to $\overline{DTRREQ}$ (REQUEST) inactive high delay. Unlike the $\overline{READYREQ}$ signal, $\overline{DTRREQ}$ does not immediately go inactive after the requested DMA transfer begins. Instead, the $\overline{DTRREQ}$ remains active for a maximum of 5 tCY + 300 ns. This delayed request pulse could trigger a second DMA transfer. To avoid this undesirable condition, a D Flip Flop is implemented to reset the $\overline{DTRREQ}$ signal inactive low following the initiation of the requested DMA transfer. To determine if back to back DMA transfers are required in a source synchronized configuration, the 80186 DMA controller samples the service request line 25 ns before T1 of the deposit cycle, the second cycle of the transfer.

* 4 Tclcl − Tclcsv(max) − U4Tpd(max) − Tdrqcl(min)

= 500 − 66 − 10.5 − 25 = 398.5 ns margin

2. tRRI: 82530 $\overline{RD}$ active low to $\overline{REQ}$ inactive high delay. Assuming source synchronized DMA transfer, the 80186 requires only one wait state to meet the tRRI spec of 200 ns. Two are included for consistency with tWRI.

* 2 Tclcl + 2($\overline{Tclcl}$wait state) − Tclrl(max) − 2(U2) Tpd(max) − Tdrqcl − tRRI

=2(125) + 2(125) − 70 − 2(5.5) − 200 = 219 ns margin

3. tWRI: 82530 $\overline{WR}$ active low to $\overline{REQ}$ inactive high delay. Assuming destination synchronized DMA transfers, the 80186 needs two wait states to meet the tWRI spec. This is because the 80186 DMA controller samples requests two clocks before the end of the deposit cycle. This leaves only 1 Tclcl + n(wait states) minus $\overline{WR}$ active delay for the 82530 to inactivate its $\overline{REQ}$ signal.

* Tclcl + 2($\overline{Tclcl}$wait state) − Tcvctv(min) − Tpd$\overline{WR}$186−$\overline{WR}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] − Tdrqcl − tWRI

=375 − 5 − [125 − 5 + 4.5 + 9.2] − 25 − 200 = 11.3 ns margin

NOTE:
If one wait state DMA interface is required, external logic, like that used on the $\overline{DTRREQ}$ signal, can be used to force the 82530 $\overline{REQ}$ signal inactive.

4. **tREC:** CLK recovery time. Due to the internal data path, a recovery period is required between SCC bus transactions to resolve metastable conditions internal to the SCC. The DMA request lines are marked from requesting service until after the tREC has elapsed. In addition, the CPU should not be allowed to violate this recovery period when interleaving DMA transfers and CPU bus cycles. Software drivers or external logic should orchestrate the CPU and DMA controller operation to prevent tREC violation.

Reset Operation

During hardware reset, the system RESET signal is asserted high for a minimum of four 80186 clock cycles (1000 ns). The 82530 requires $\overline{WR}$ and $\overline{RD}$ to be simultaneously asserted low for a minimum of 250 ns.

* 4 Tclcl − U3 Tpd(max) − 2(U2) Tpd(max) + U4 Tpd(min) − tREC

= 1000 − 17.5 − 2(5.5) + 3.5 − 250 ns = 725 ns margin

intel®

APPLICATION NOTE

AP-236

November 1986

Implementing StarLAN with the Intel 82588

ADI GOLBERT
DATA COMMUNICATIONS OPERATION

SHARAD GANDHI
FIELD APPLICATIONS-EUROPE

Order Number: 231422-003

1.0 INTRODUCTION

Personal computers have become the most prolific workstation in the office, serving a wide range of needs such as word processing, spreadsheets, and data bases. The need to interconnect PCs in a local environment has clearly emerged, for purposes such as the sharing of file, print, and communication servers; downline loading of files and application programs; electronic mail; etc. Proliferation of the PC makes it the workstation of choice for accessing the corporate mainframe/s; this function can be performed much more efficiently and economically when clusters of PCs are already interconnected through Local Area Networks (LANs). According to market surveys, the installed base of PCs in business environments reached about 10 million units year-end '85, with only a small fraction connected via LANs. The installed base is expected to double by 1990. There is clearly a great need for locally interconnecting these machines; furthermore, end users expect interconnectability across vendors. Thus, there is an urgent need for industry standards to promote cost effective PC LANs.

A large number of proprietary PC LANs have become available for the office environment over the past several years. Many of these suffer from high installed cost, technical deficiencies, non-conformance to industry standards, and general lack of industry backing. StarLAN, in Intel's opinion, is one of the few networks which will emerge as a standard. It utilizes a proven network access method, it is implemented with proven VLSI components; it is cost effective, easily installable and reconfigurable; it is technically competent; and it enjoys the backing of a large cross section of the industry which is collaborating to develop a standard (IEEE 802.3, type 1BASE5).

1.1 StarLAN

StarLAN is a 1 Mb/s network based on the CSMA/CD access method (Carrier Sense, Multiple Access with Collision Detection). It works over standard, unshielded, twisted pair telephone wiring. Typically, the wiring connects each desk to a wiring closet in a star topology (from which the IEEE Task Force working on the standard derived the name StarLAN in 1984). In fact, telephone and StarLAN wiring can coexist in the same twisted pair bundle connecting a desk to the wiring closet. Abundant quantities of unused phone wiring exist in most office environments, particularly in the U.S. The StarLAN concept of wiring and networking concepts was originated by AT&T Information Systems.

1.2 The 82588

The 82588 is a single-chip LAN controller designed for CSMA/CD networks. It integrates in one chip all functions needed for such networks. Besides inplementing the standard CSMA/CD functions like framing, deferring, backing off and retrying on collisions, transmitting and receiving frames, it performs data encoding and decoding in Manshester or NRZI format, carrier sensing and collision detection, all up to a speed of 2 Mb/s (independent of the chosen encoding scheme). These functions make it an optimum controller for a StarLAN node. The 82588 has a very conventional microcomputer bus interface, easing the job of interfacing it to any processor.

1.3 Organization of the Application Note

This application note has two objectives. One is to describe StarLAN in practical terms to prospective implementers. The other is to illustrate designing with 82588, particularly as related to StarLAN which is expected to emerge as its largest application area.

Section 2 of this Application Note describes the StarLAN network, its basic components, collision detection, signal propagation and network parameters. Sections 3 and 4 describe the 82588 LAN controller and its role in the StarLAN network. Section 5 goes into the details of designing a StarLAN node for the IBM PC. Section 6 describes the design of the HUB. Both these designs have been implemented and operated in an actual StarLAN environment. Section 7 documents the software used to drive the 82588. It gives the actual procedures used to do operations like, configure, transmit and receive frames. It also shows how to use the DMA controller and interrupt controller in the IBM PC and goes into the details of doing I/O on the PC using DOS calls. Appendix A shows oscilloscope traces of the signals at various points in the network. Appendix B describes the multiple point extension (MPE) being considered by IEEE. Appendixes C and D talk about advanced usages of the 82588; working with only one DMA channel, and measuring network delays with the 82588.

1.4 References

For additional information on the 82588, see the Intel Microcommunications Handbook. StarLAN specification are currently available in draft standard form through the IEEE 802.3 Working Group.

2.0 StarLAN

StarLAN is a low cost 1 Mb/s networking solution aimed at office automation applications. It uses a star

topology with the nodes connected in a point-to-point fashion to a central HUB. HUBs can be connected in a hierarchical fashion. Up to 5 levels are supported. The maximum distance between a node and the adjacent HUB or between two adjacent HUBs is 800 ft. (about 250 meters) for 24 gauge wire and 600 ft. (about 200 meters) for 26 gauge wire. Maximum node-to-node distance with one HUB is 0.5 km, hence IEEE 802.3 designation of type 1BASE5. 1 stands for 1 Mb/s and BASE for baseband. (StarLAN doesn't preclude the use of more than 800 ft wiring provided 6.5 dB maximum attenuation is met, and cable propagation delay is no more than 4 bit times).

One of the most attractive features of StarLAN is that it uses telephone grade twisted pair wire for the transmission medium. In fact, existing installed telephone wiring can also be used for StarLAN. Telephone wiring is very economical to buy and install. Although use of telephone wiring is an obvious advantage, for small clusters of nodes, it is possible to work around the use of building wiring.

Factors contributing to low cost are:

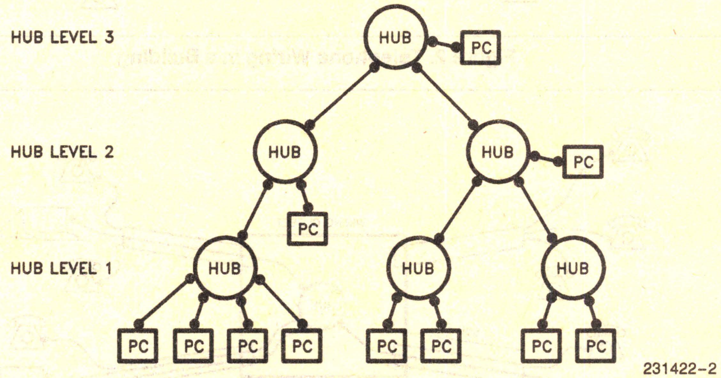
1) Use of telephone grade, unshielded, 24 or 26 gauge twisted pair wire transmission media.
2) Installed base of redundant telephone wiring in most buildings.
3) Buildings are designed for star topology wiring. They have conduits leading to a central location.
4) Availability of low cost VLSI LAN controllers like the 82588 for low cost applications and the 82586 for high performance applications.
5) Off-the-shelf, Low cost RS-422, RS-485 drivers/receivers compatible with the StarLAN analog interface requirements.

2.1 StarLAN Topology

StarLAN, as the name suggests, uses a star topology. The nodes are at the extremities of a star and the central point is called a HUB. There can be more than one HUB in a network. The HUBs are connected in a hierarchical fashion resembling an inverted tree, as shown in Figure 1, where nodes are shown as PCs. The HUB at the base (at level 3) of the tree is called the Header Hub (HHUB) and others are called Intermediate HUBs (IHUB). It will become apparent, later in this section, that topologically, this entire network of nodes and HUBs is equivalent to one where all the nodes are connected to a single HUB. Also StarLAN doesn't limit the number of nodes or HUBS at any given level.

2.1.1 TELEPHONE NETWORK

StarLAN is structured to run parallel to the telephone network in a building. The telephone network has, in fact, exactly the same star topology as StarLAN. Let us now examine how the telephone system is typically laid out in a building in the USA. Figure 2 shows how a typical building is wired for telephones. 24 gauge unshielded twisted pair wires emanate from a Wiring Closet. The wires are in bundles of 25 or 50 pairs. The bundle is called D inside wiring (DIW). The wires in these cables end up at modular telephone jacks in the wall. The telephone set is either connected directly to



*Maximum of 5 HUB levels.
*PCs or DTEs can connect directly at any level.

Figure 1. StarLAN Topology

the jack or through an extension cable. Each telephone generally needs one twisted pair for voice and another for auxilliary power. Thus, each modular jack has 2 twisted pairs (4 wires) connected to it. A 25 pair DIW cable can thus be used for up to 12 telephone connections. In most buildings, not all pairs in the bundle are used. Typically, a cable is used for only 4 to 8 telephone connections. This practice is followed by telephone companies because it is cheaper to install extra wires initially, rather than retrofitting to expand the existing number of connections. As a result, a lot of extra, unused wiring exists in a building. The stretch of cable between the wiring closet and the telephone jack is typically less than 800 ft. (250 meters). In the wiring closet the incoming wires from the telephones are routed to another wiring closet, a PABX or to the central office through an interconnect matrix. Thus, the wiring closet is a concentration point in the telephone network. There is also a redundancy of wires between the wiring closets.

2.1.2 StarLAN AND THE TELEPHONE NETWORK

StarLAN does not have to run on building wiring, but the fact that it can significantly adds to its attractiveness. Figure 3 shows how StarLAN piggybacks on telephone wiring. Each node needs two twisted pair wires to connect to the HUB. The unused wires in the 25 pair DIW cables provide an electrical path to the wiring closet, where the HUB is located. Note that the telephone and StarLAN are electrically isolated. They only use the wires in the same bundle cable to connect to the wiring closet. Within the wiring closet, StarLAN wires connect to a HUB and telephone wires are routed to a different path. Similar cable sharing can occur in connecting HUBs to one another. See Figure 4 for a typical office wired for StarLAN through telephone wiring.

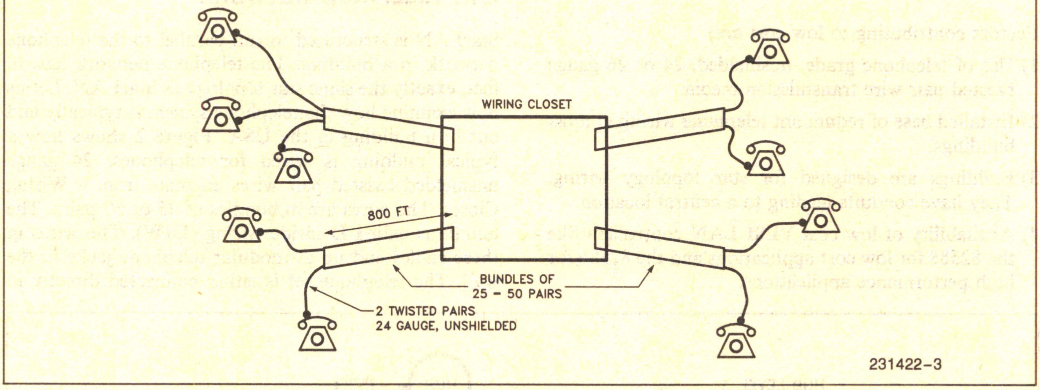
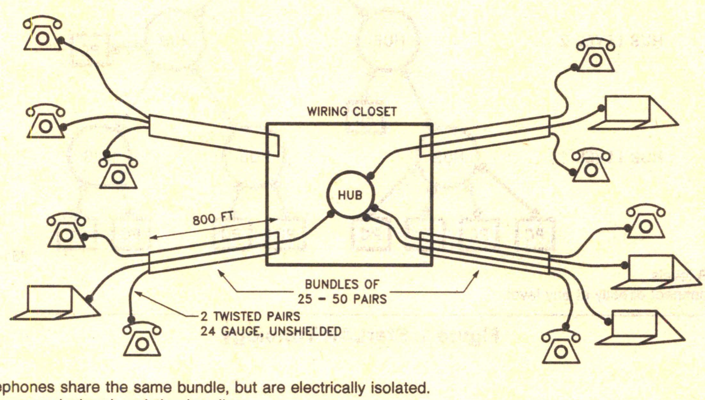


Figure 2. Telephone Wiring in a Building



*StarLAN and telephones share the same bundle, but are electrically isolated.
*StarLAN uses the unused wires in existing bundles.

Figure 3. Coexistence of Telephone and StarLAN

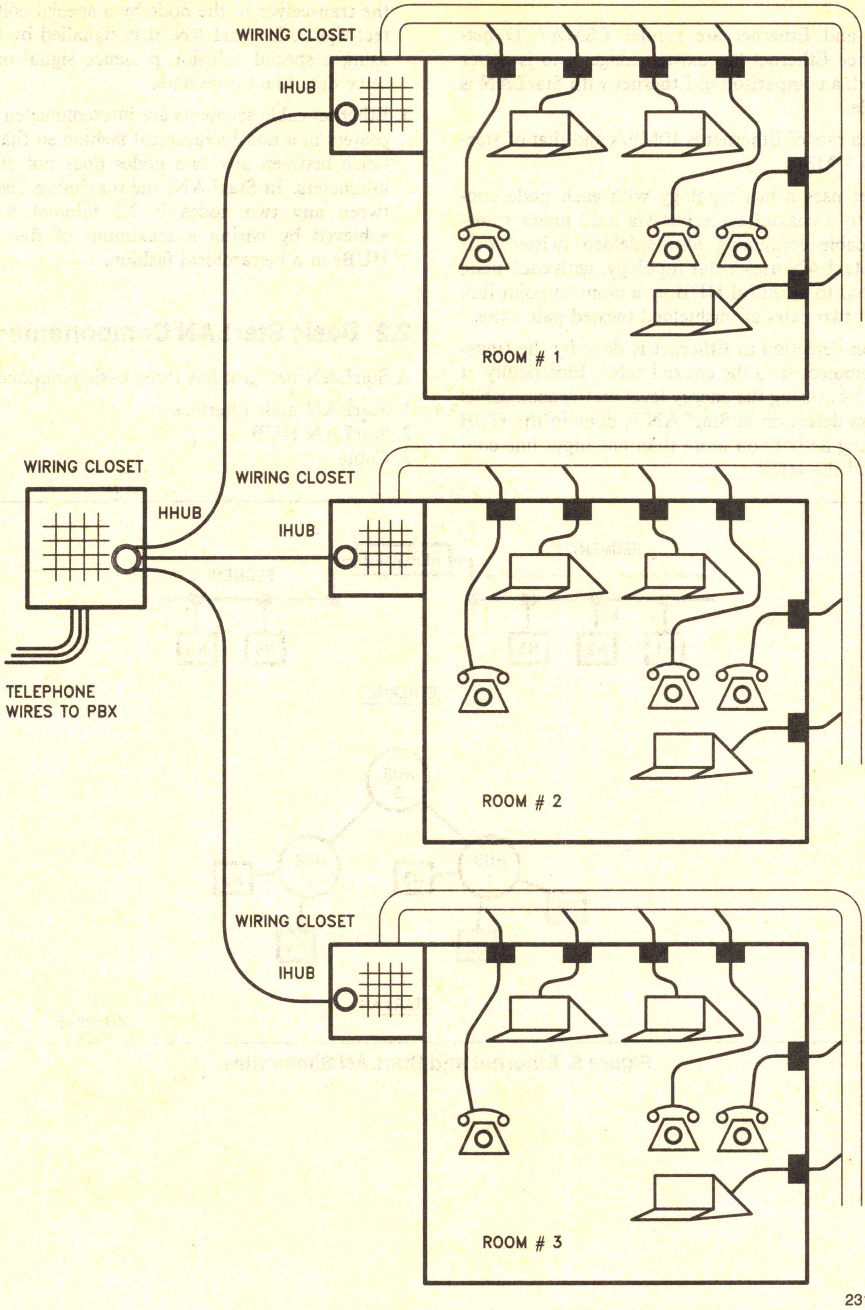


Figure 4. A Typical Office Using Telephone Wiring for StarLAN

2.1.3 StarLAN AND Ethernet

StarLAN and Ethernet are similar CSMA/CD networks. Since Ethernet has existed longer and is better understood, a comparison of Ethernet with StarLAN is worthwhile.

1. The data rate of Ethernet is 10Mb/s and that of StarLAN is 1 Mb/s.
2. Ethernet uses a bus topology with each node connected to a coaxial cable bus via a 50 meter transceiver cable containing four shielded twisted pair wires. StarLAN uses a star topology, with each node connected to a central HUB by a point to point link through two pairs of unshielded twisted pair wires.
3. Collision detection in Ethernet is done by the transceiver connected to the coaxial cable. Electrically, it is done by sensing the energy level on the coax cable. Collision detection in StarLAN is done in the HUB by sensing activity on more than one input line connected to the HUB.
4. In Ethernet, the presence of collision is signalled by the transceiver to the node by a special collision detect signal. In StarLAN, it is signalled by the HUB using a special collision presence signal on the receive data line to the node.
5. Ethernet cable segments are interconnected using repeaters in a non-hierarchical fashion so that the distance between any two nodes does not exceed 2.8 kilometers. In StarLAN, the maximum distance between any two nodes is 2.5 kilometers. This is achieved by wiring a maximum of five levels of HUBs in a hierarchical fashion.

2.2 Basic StarLAN Components

A StarLAN network has three basic components:

1. StarLAN node interface
2. StarLAN HUB
3. Cable

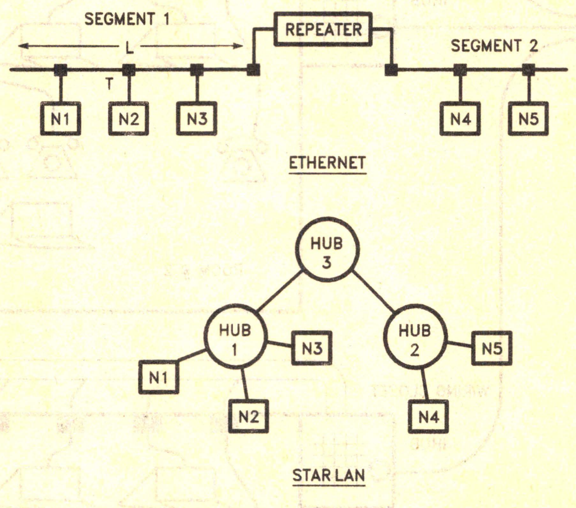


Figure 5. Ethernet and StarLAN Similarities

2.2.1 A StarLAN NODE INTERFACE

Figure 6 shows a typical StarLAN node interface. It interfaces to a processor on the system side. The processor runs the networking software. The heart of the node interface is the LAN controller which does the job of receiving and transmitting the frames in adherence to the IEEE 802.3 standard protocol. It maintains all the timings—like the slot time, interframe spacing etc.—required by the network. It performs the functions of framing, deferring, backing-off, collision detection which are necessary in a CSMA/CD network. It also does Manchester encoding of data to be transmitted and clock separation—or decoding—of the Manchester encoded data that is received. These signals before going to the unshielded twist pair wire, may undergo pulse shaping (optional) pulse shaping basically slows down the fall/rise times of the signal. The purpose of that is to diminish the effects of cross-talk and radiation on adjacent pairs sharing the same bundle (digital voice, T1 trunks, etc). The shaped signal is sent on to the twisted pair wire through a pulse transformer for DC isolation. The signals on the wire are thus differential, DC isolated from the node and almost sinusoidal (due to shaping and the capacitance of the wire).

NOTE:
Work done by the IEEE 802.3 committee has shown that no slew rate control on the drivers is required. Shaping by the transformer and the cable is sufficient to avoid excessive EMI radiation and crosstalk.

The squelch circuit prevents idle line noise from affecting the receiver circuits in the LAN controller. The squelch circuit has a 600 mv threshold for that purpose. Also as part of the squelch circuitry an envelope detector is implemented. Its purpose is to generate an envelope of the transitions of the RXD line. Its output serve as a carrier sense signal. The differential signal from the HUB is received using a zero-crossing RS-422 receiver. Output of the receiver, qualified by the squelch circuit, is fed to the RxD pin of the LAN controller. The RxD signal provides three kinds of information:

1) Normal received data, when receiving the frame.
2) Collision information in the form of the collision presence signal from the HUB.
3) Carrier sense information, indicating the beginning and the end of frame. This is useful during transmit and receive operations.

2.2.2 StarLAN HUB

HUB is the point of concentration in StarLAN. All the nodes transmit to the HUB and receive from the HUB. Figure 7 shows an abstract representation of the HUB. It has an upstream and a downstream signal processing unit. The upstream unit has N signal inputs and 1 signal output. And the downstream unit has 1 input and N output signals. The inputs to the upstream unit come from the nodes or from the intermediate HUBs (IHUBs) and its output goes to a higher level HUB. The downstream unit is connected the other way around; input from an upper level HUB and the outputs to nodes or lower level IHUBs. Physically each input and output consist of one twisted pair wire carrying a differential signal. The downstream unit essentially just re-times the signal received at the input, and sends it to all its outputs. The functions performed by the upstream unit are:

1. Collision detection
2. Collision Presence signal generation
3. Signal Retiming
4. Jabber Function
5. Start of Idle protection timer

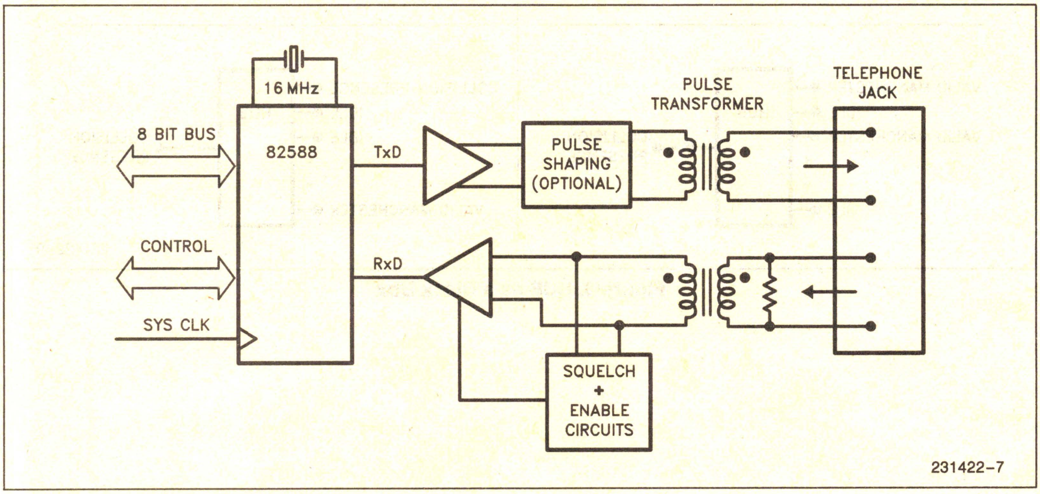


Figure 6. 82588 Based StarLAN Node

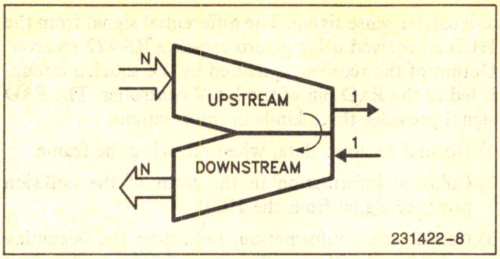


Figure 7. A StarLAN HUB

The collision detection in the HUB is done by sensing the activity on the inputs. If there is activity (or transitions) on more than one input, it is assumed that more than one node is transmitting. This is a collision. If a collision is detected, a special signal called the Collision Presence Signal is generated. This signal is generated and sent out as long as activity is sensed on any of the input lines. This signal is interpreted by every node as an occurrence of collision. If there is activity only on one input, that signal is re-timed—or cleaned up of any accumulated jitter—and sent out. Figure 8 shows the input to output relations of the HUB as a black box.

If a node transmits for too long the HUB exercises a Jabber function to disable the node from interfering with traffic from other nodes. There are two timers in the HUB associated with this function and their operation is described in section 6.

The last function implemented by the HUB is the start of Idle protection timer. During the end of reception, the HUB will see a long undershoot at its input port. This undershoot is a consequence of the transformer discharging accumulated charge during the 2 microseconds of high of the idle pattern. The HUB should implement a protection mechanism to avoid the undesirable effects of that undershoot.

Figure 9 shows a block diagram of the HUB. A switch position determines whether the HUB is an IHUB or a HHUB (Header HUB). If the HUB is an IHUB, the switch decouples the upstream and the downstream units. HHUB is the highest level HUB; it has no place to send its output signal, so it returns its output signal (through the switch) to the outputs of the downstream unit. There is one and only one HHUB in a StarLAN network and it is always at the base of the tree. The returned signal eventually reaches every node in the network through the intermediate nodes (if any). StarLAN specifications do not put any restrictions on the number of IHUBS at any level or on number of inputs to any HUB. The number of inputs per HUB are typically 6 to 12 and is dictated by the typical size of clusters in a given networking environment.

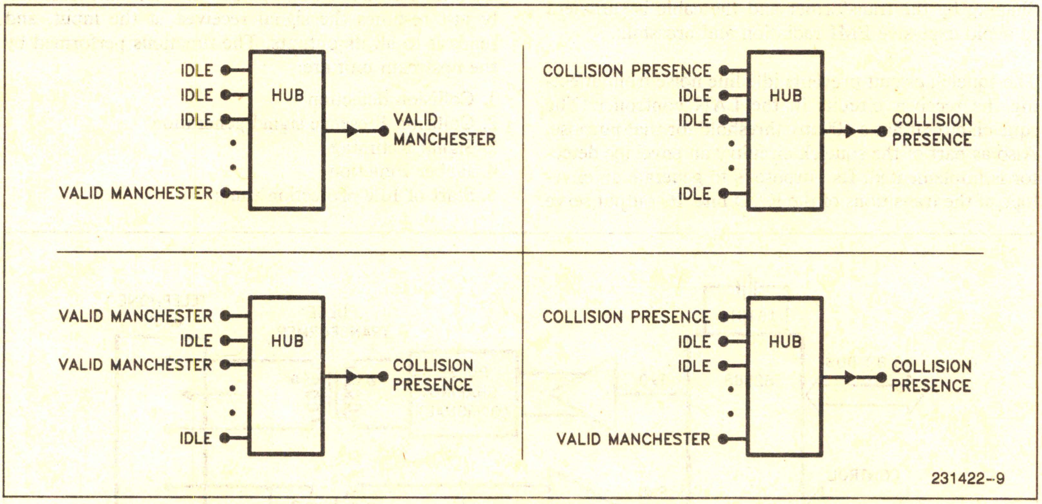


Figure 8. HUB as a Black Box

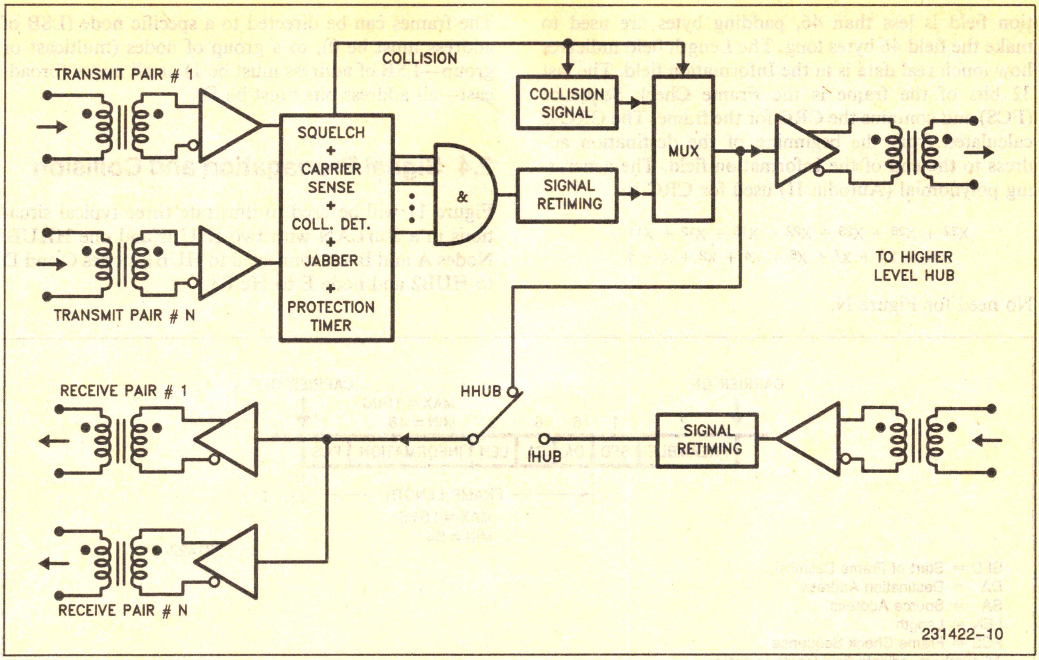


Figure 9. StarLAN HUB Block Diagram

2.2.3 StarLAN CABLE

Unshielded telephone grade twisted pair wires are used to connect a node to a HUB or to connect two HUBs. This is one of the cheapest types of wire and an important factor in bringing down the cost of StarLAN.

Although the 24 gauge wire is used for long stretches, the actual connection between the node and the telephone jack in the wall is done using extension cable, just like connecting a telephone to a jack. For very short StarLAN configurations, where all the nodes and the HUB are in the same room, the extension cable with plugs at both ends may itself be sufficient for all the wiring. (Extension cables must be of the twisted pair kind, no flat cables are allowed).

The telephone twisted pair wire of 24 gauge has the following characteristics:

Attenuation	: 42.55 db/mile @ 1 MHz
DC Resistance	: 823.69 Ω/mile
Inductance	: 0.84 mH/mile
Capacitance	: 0.1 μF/mile
Impedance	: 92.6Ω, −4 degrees @ 1 MHz

Experiments have shown that the sharing of the telephone cable with other voice and data services does not cause any mutual harm due to cross-talk and radiation, provided every service meets the FCC limits.

Although it is outside the scope of the IEEE 802.3 1BASE5 standard, there is considerable interest in using fiber optics and coaxial cable for node to HUB or HUB to HUB links especially in noisy and factory environments. Both these types of cables are particularly suited for point-to-point connections. Even mixing of different types of cables is possible (this kind of environments are not precluded).

NOTE:
StarLAN IEEE 802.3 1BASE5 draft calls for a maximum attenuation of 6.5 dB between the transmitter and the corresponding receiver at all frequencies between 500 KHz to 1 MHz. Also the maximum allowed cable propagation delay is 4 microseconds.

2.3 Framing

Figure 10 shows the format of a 802.3 frame. The beginning of the frame is marked by the carrier going active and the end marked by carrier going inactive. The preamble has a 56 bit sequence of 101010 . . . . ending in a 0. This is followed by 8 bits of start of frame delimiter (sfd) — 10101011. These bits are transmitted with the MSB (leftmost bit) transmitted first. Source and destination fields are 6 bytes long. The first byte is the least significant byte. These fields are transmitted with LSB first. The length field is 2 bytes long and gives the length of data in the Information field. The entire information field is a minimum of 46 bytes and a maximum of 1500 bytes. If the data content of the Informa-

tion field is less than 46, padding bytes are used to make the field 46 bytes long. The Length field indicates how much real data is in the Information field. The last 32 bits of the frame is the Frame Check Sequence (FCS) and contains the CRC for the frame. The CRC is calculated from the beginning of the destination address to the end of the Information field. The generating polynomial (Autodin II) used for CRC is:

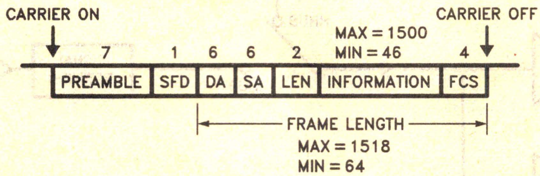
$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

No need for Figure N.

The frames can be directed to a specific node (LSB of address must be 0), to a group of nodes (multicast or group—LSB of address must be 1) or all nodes (broadcast—all address bits must be 1).

2.4 Signal Propagation and Collision

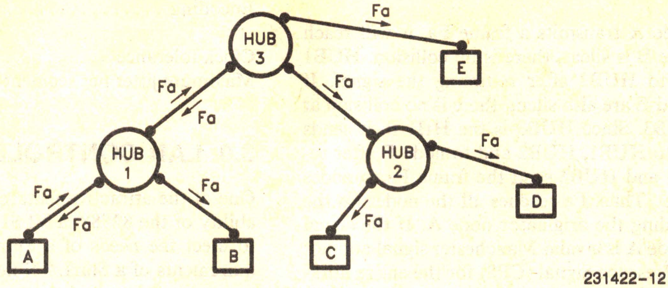
Figure 11 will be used to illustrate three typical situations in a StarLAN with two IHUBs and one HHUB. Nodes A and B are connected to HUB1, nodes C and D to HUB2 and node E to HUB3.



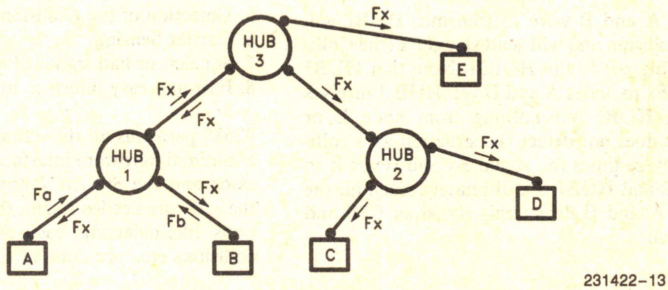
231422-11

SFD = Start of Frame Delimiter
DA = Destination Address
SA = Source Address
LEN = Length
FCS = Frame Check Sequence
All numbers indicate field length in octets.

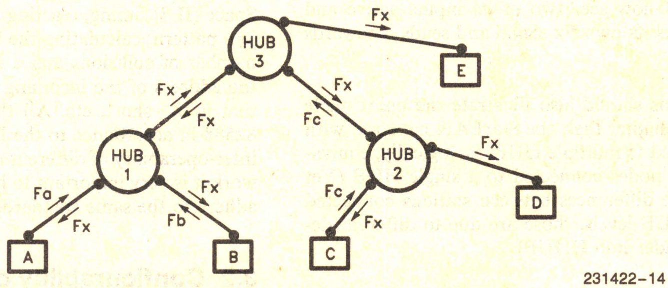
Figure 10. Framing



Situation #1. A Transmitting



Situation #2. A & B Transmitting



Situation #3. A, B & C Transmitting

HUB1, HUB2 are IHUBs
HUB3 is the HHUB
Fa, Fb, Fc—Frames from nodes A, B & C
Fx—Collision Presence Signal

Figure 11. Signal Propagation and Collisions

2.4.1 Situation #1

Whenever node A transmits a frame Fa, it will reach HUB1. If node B is silent, there is no collision. HUB1 will send Fa to HUB3 after re-timing the signal. If nodes C, D and E are also silent, there is no collision at HUB2 or HUB3. Since HUB3 is the HHUB, it sends the frame Fa to HUB1, HUB2 and to node E after re-timing. HUB1 and HUB2 send the frame Fa to nodes A, B and C, D. Thus, Fa reaches all the nodes on the network including the originator node A. If the signal received by node A is a valid Manchester signal and not the Collision Presence Signal (CPS) for the entire duration of the slot time, then the node A assumes that it was a successful transmission.

2.4.2 Situation #2

If both nodes A and B were to transmit, HUB1 will detect it as a collision and will send signal Fx (the Collision Presence Signal) to the HUB3—Note that HUB1 does not send Fx to nodes A and B yet. HUB 3 receives a signal from HUB1 but nothing from node E or HUB2, thus it does not detect the situation as a collision and simply re-times the signal Fx and sends it to node E, HUB2 and HUB1. Fx ultimately reach all the nodes. Nodes A and B detect this signal as CPS and call it a collision.

2.4.3 Situation #3

In addition to nodes A and B, if node C were also to transmit, the situation at HUB1 will be the same as in situation #2. HUB2 will propagate Fc from C towards HUB3. HUB3 now sees two of its inputs active and hence generates its own Fx signal and sends it towards each node.

These situations should also illustrate the point made earlier in the chapter that, the StarLAN network, with nodes connected to multiple HUBs is, logically, equivalent to all the nodes connected to a single HUB (Yet there are some differences between stations connected at different HUB levels, those are due to different delays to the header hub HHUB).

2.5 StarLAN System and Network Parameters

Preamble length (incl. sfd)	64 bits
Address length	6 bytes
FCS length CRC (Autodin II)	32 bits
Maximum frame length	1518 bytes
Minimum frame length	64 bytes
Slot time	512 bit times
Interframe spacing	96 bit times
Minimum jam timing	32 bit times
Maximum number of collisions	16
Backoff limit	10
Backoff method	Truncated binary exponential
Encoding	Manchester
Clock tolerance	±0.01% (100 ppm)
Maximum jitter per segment	±62.5 ns

3.0 LAN CONTROLLER FOR StarLAN

One of the attractive features of StarLAN is the availability of the 82588, a VLSI LAN controller, designed to meet the needs of a StarLAN node. The main requirements of a StarLAN node controller are:

1. IEEE 802.3 compatible CSMA/CD controller.
2. Configurable to StarLAN network and system parameters.
3. Generation of all necessary clocks and timings.
4. Manchester data encoding and decoding.
5. Detection of the Collision Presence Signal.
6. Carrier Sensing.
7. Squelch or bad signal filtering.
8. Fast and easy interface to the processor.

82588 performs all these functions in silicon, providing a minimal hardware interface between the system processor and the StarLAN physical link. It also reduces the software needed to run the node, since a lot of functions, like deferring, back off, counting the number of collisions etc., are done in silicon.

3.1 IEEE 802.3 Compatibility

The CSMA/CD control unit on the 82588 performs the functions of deferring, maintaining the Interframe Space (IFS) timing, reacting to collision by generating a jam pattern, calculating the back-off time based on the number of collisions and a random number, decoding the address of the incoming frame, discarding a frame that is too short, etc. All these are performed by the 82588 in accordance to the IEEE 802.3 standards. For inter-operability of different nodes on the StarLAN network it is very important to have the controllers strictly adhere to the same standards.

3.2 Configurability of the 82588

Almost all the networking parameters are programmable over a wide range. This means that the StarLAN parameters form a subset of the total potential of the 82588. This is a major advantage for networks whose standards are being defined and are in a flux. It is also an advantage when carrying over the experience gained with the component in one network to other applications, with differing parameters (leveraging the design).

The 82588 is initialized or configured to its working environment by the CONFIGURE command. After the execution of this command, the 82588 knows its system and network parameters. A configure block in

memory is loaded into the 82588 by DMA. This block contains all the parameters to be programmed as shown in Figure 12. Following is a partial list of the parameters with the programmable range and the StarLAN value:

Parameter	Range	StarLAN Value
Preamble length	2, 4, 8, 16 bytes	8
Address length	0 to 6 bytes	6
CRC type	16, 32 bit	32
Minimum frame length	6 to 255 bytes	64
Interframe spacing	12 to 255 bit times	96
Slot time	1 to 2047 bit times	512
Number of retries	0 to 15	15

Parameter	Range	StarLAN Value
Data encoding	NRZI, Man., Diff. Man.	Manch.
Collision detection	Code viol., Bit comp.	Code Viol.

Beside these, there are many other options available, which may or may not apply to StarLAN:

Data sampling rate of 8 or 16
Operating in Promiscuous mode
Reception of Broadcast frames
Internal loopback operation
External loopback operation
Transmit without CRC
HDLC Framing
:
:

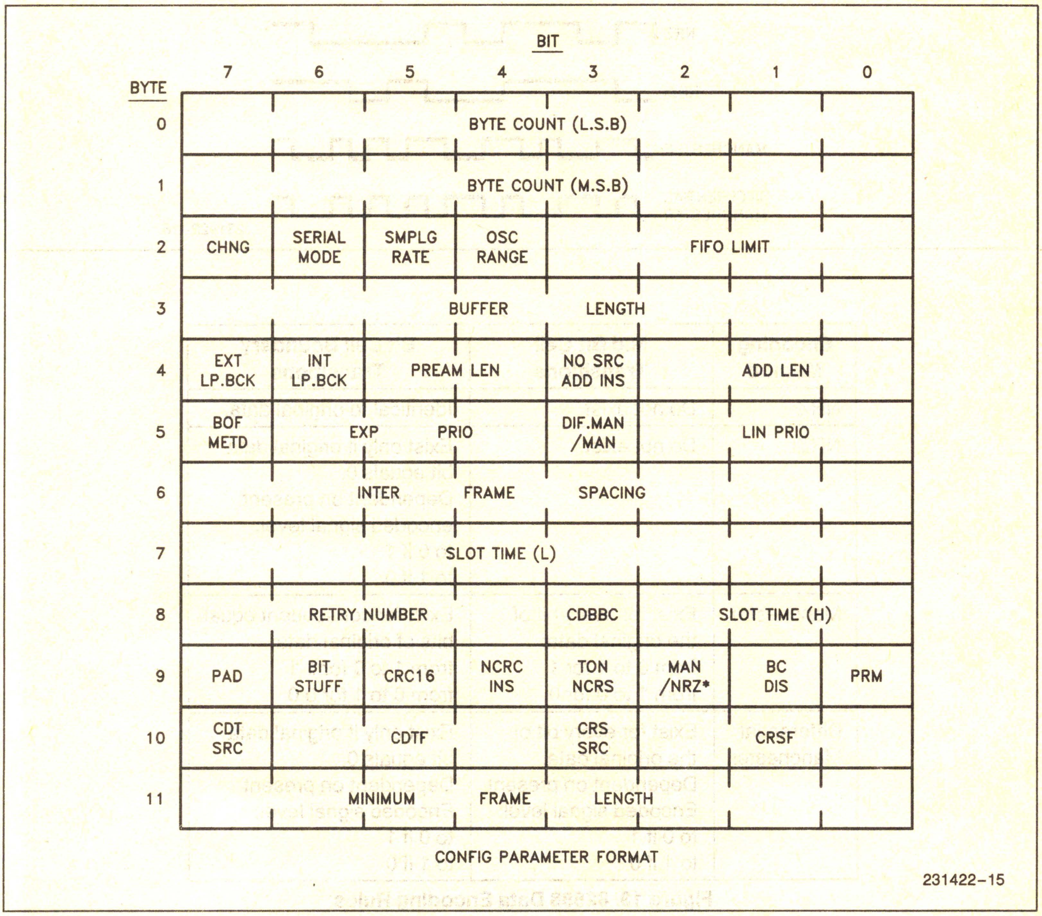


Figure 12. Configuration Block

3.3 Clocks and Timers

The 82588 requires two clocks; one for the operation of the system interface and another for the serial side. Both clocks are totally asynchronous to each other. This permits transmitting and receiving frames at data rates that are virtually independent of the speed at which the system interface operates.

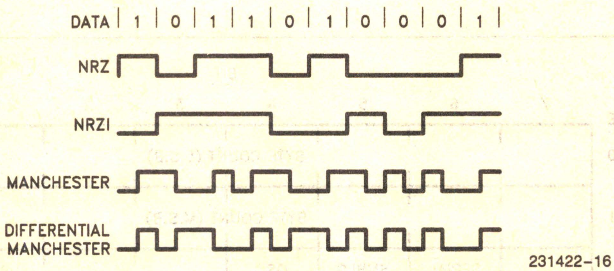
The serial clock can be generated on chip using just an external crystal of a value 8 or 16 times the desired bit rate. An external clock may also be used.

The 82588 has a set of timers to maintain various timings necessary to run the CSMA/CD control unit. These are timings for the Slot time, Interframe spacing time, Back off time, Number of collisions, Minimum frame length, etc. These timers are started and stopped automatically by the 82588.

3.4 Manchester Data Encoding and Decoding

In StarLAN the data transmitted by the node must be encoded in Manchester format. The node should also be able to decode Manchester encoded data when receiving a frame—a process also known as clock recovery. The 82588 does the encoding and decoding of data bits on chip for data rates up to 2 Mb/s.

Besides Manchester, the 82588 can also do encoding and decoding in NRZI and Differential Manchester formats. Figure 13 shows samples of encoding in



Encoding Method	Mid Bit Cell Transitions	Bit Cell Boundary Transitions
NRZ	Do not exist.	Identical to original data.
NRZI	Do not exist.	Exist only if original data bit equals 0. Dependent on present encoded signal level: to 0 if 1 to 1 if 0
Manchester	Exist for every bit of the original data: from 0 to 1 for 1 from 1 to 0 for 0	Exist for consequent equal bits of original data: from 1 to 0 for 1 1 from 0 to 1 for 0 0
Differential Manchester	Exist for every bit of the original data. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0	Exist only if original data bit equals 0. Dependent on present Encoded signal level: to 0 if 1 to 1 if 0

Figure 13. 82588 Data Encoding Rules

these three formats. The main advantage of NRZI over the other two is that NRZI requires half the channel bandwidth, for any given data rate. On the other hand, since the NRZI signal does not have as many transitions as the other two, clock recovery from it is more difficult. The main advantage of Differential Manchester over straight Manchester is that for a signal that is differentially driven (as in RS 422), crossing of the two wires carrying the data does not change the data received at the receiver. In other words, NRZI and Differential Manchester encoding methods are polarity insensitive (Even though NRZI, Differential Manchester are polarity insensitive, the 82588 expects a high level in the RXD line to detect carrier inactive at the end of frames).

3.5 Detection of the Collision Presence Signal

In a StarLAN network, HUB informs the nodes that a collision has occurred by sending the Collision Presence Signal (CPS) to the nodes. The CPS signal is a special signal which contains violations in Manchester encoding. Figure 14 shows the CPS signal. It has a 5 ms period, looking very much like a valid Manchester signal except for missing transitions (or violations) at periodic intervals. When the 82588 decodes this signal, it fails to see mid-cell transitions repeatedly at intervals of 2.5 bit times and hence calls it a code violation. The edges of CPS are marked for illustration as a, b, c, d, . . . l. Let us see how the 82588 interprets the signal if it starts calling the edge 'a' as the mid-cell transition for '1'. Then edge at 'b' is '0'. Now the 82588 expects to see an edge at '*' but since there is none, it is a Manchester code violation. The edge that eventually does occur at 'd' is then used to re-synchronize and, since it is a falling edge, it is taken as a mid-cell transition for '0'. The edge at 'e' is for a '1' and then again there is no edge at '*'. This goes on, with the 82588 flagging code violation and re-synchronizing again every 2.5 bit times. When a transmitting node sees this CPS signal being returned by the HUB (instead of a valid Manchester signal it transmitted), it assumes that a collision occurred. The 82588 has two built-in mechanisms to detect collisions. These mechanisms are very general and can be used for a very broad class of applications to detect collisions in a CSMA/CD network. Using these mechanisms, the 82588 can detect collisions (two or more nodes transmitting simultaneously) by just receiving the collided signal during transmission, even if there was no HUB generating the CPS signal.

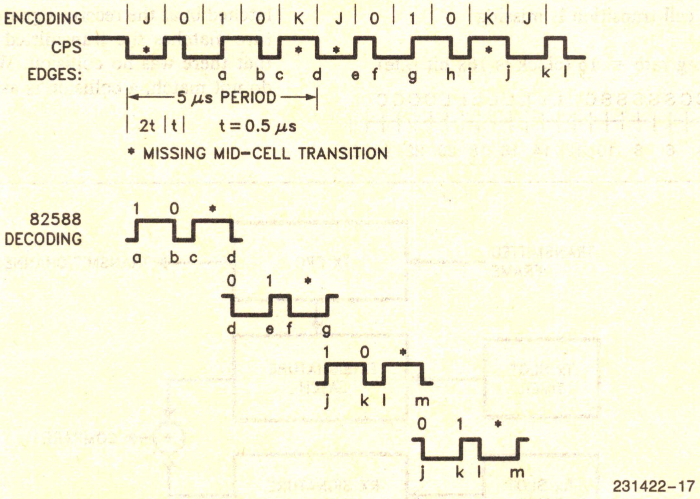


Figure 14. 82588 Decoding the Collision Presence Signal

3.5.1 COLLISION DETECTION BY CODE VIOLATION

If during transmission, the 82588 sees a violation in the encoding (Manchester, NRZI or Differential Manchester) used, then it calls it a collision by aborting the transmission and transmitting a 32 bit jam pattern. The algorithm used to detect collisions, and to do the data decoding, is based on finding the number of sampling clocks between an edge to the next one. Suppose an edge occurred at time 0, the sampling instant of the next edge determines whether it was a collision (C), a long pulse (L)—with a nominal width of 1 bit time—,or a short pulse (S)—nominal width of half a bit time. The following two charts show the decoding and collision detection algorithm for sampling rates of 8 and 16 when using Manchester encoding. The numbers at the bottom of the line indicate sampling instances after the occurrence of the last edge (at 0). The alphabets on the top show what would be inferred by the 82588 if the next edge were to be there.

Sampling rate = 8 (clock is 8x bit rate)

C C S S S L L L L L C C

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Collision also if:
RxD stays low for 13 samples or more
A mid cell transition is missing

Sampling rate = 16 (clock is 16x bit rate)

C C C C C S S S S S C L L L L L L L L L C C C C

0 2 4 6 8 10 12 14 16 18 20 22 24 26

Collision also if:
RxD stays low for 25 samples or more
A mid cell transition is missing

A single instance of code violation can qualify as collision. The 82588 has a parameter called collision detect filter (CDT Filter) that can be configured from 0 to 7. This parameter determines for how many bit times the violation must remain active to be flagged as a collision. For StarLAN CDT Filter must be configured to 0—that is disabled.

3.5.2 COLLISION DETECTION BY SIGNATURE (OR BIT) COMPARISON

This method of collision detection compares a signature of the transmitted data with that of the data received on the RxD pin while transmitting. Figure 15 shows a block diagram of the logic. As the frame is transmitted it flows through the CRC generation logic. A timer, called the Tx slot timer, is started at the same time that the CRC generation starts. When the count in the timer reaches the slot time value, the current value of the CRC generator is latched in as the transmit signature. As the frame is returned back (through the HUB) it flows through the CRC checker. Another timer—Rx slot timer—is started at the same time as the CRC checker starts checking. When this timer reaches the slot time value, the current value of the CRC checker is latched in as the receive signature. If the received signature matches the transmitted one, then it is assumed that there was no collision. Whereas, if the signatures do not match, a collision is assumed to have occurred.

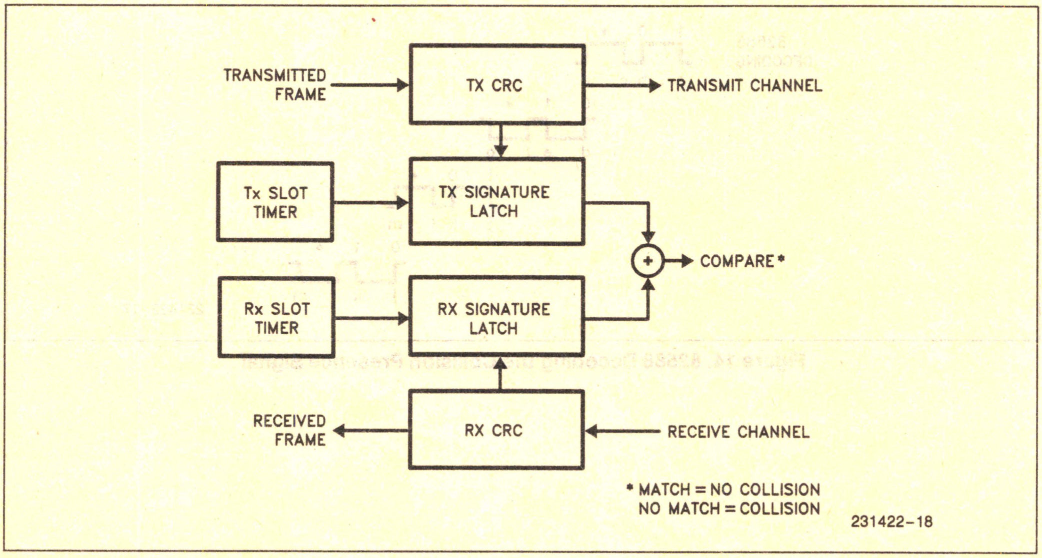


Figure 15. Collision Detection by Signature Comparison

Note that, even if the collision were to occur in the first few bits of the frame, a slot time must elapse before it is detected. In the code violation method, collision is detected within a few bit times. However, since the signature method compares the signatures, which are characteristic of the frame being transmitted, it is more robust. The code violation method can be fooled by returning a signal to the 82588 which is not the same as the transmitted signal but is a valid Manchester signal—like a 1 MHz signal. Both methods can be used simultaneously giving a combination of speed and robustness.

NOTE:

In order to reliably detect a collision using the collision by bit comparison mode, the transmitter must still be transmitting up to the point where the receiver has seen enough bits to complete its signature. Otherwise, the transmitter may be done before the RX signature is completed resulting in an undetected collision. A sufficient condition to avoid this situation is to transmit frames with a minimum length of 1.5 * slottime (see Figure 16).

3.5.3 ADDITIONAL COLLISION DETECTION MECHANISM

In addition to the collision detection mechanisms described in the preceding sections, the 82588 also flags collision when after starting a transmission any of the following conditions become valid:

a) Half a slot time elapses and the carrier sense of 82588 is not active.

b) Half a slot time + 16 bit times elapse and the opening flag (sfd) is not detected.

c) Carrier sense goes inactive after an opening flag is received with transmitter still active.

These mechanisms add a further robustness to the collision detection mechanism of the 82588. It is also possible to OR an externally generated collision detect signal to the internally generated condition by bit comparison (see Figure 17).

3.6 Carrier Sensing

A StarLAN network is considered to be busy if there are transitions on the cable. Carrier is supposed to be active if there are transitions. Every node controller needs to know when the carrier is active and when not. This is done by the carrier sensing circuitry. On the 82588 this circuit is on chip. It looks at the RxD (receive data) pin and if there are transitions, it turns on an internal carrier sense signal. It turns off the carrier sense signal if RxD remains in idle (high) state for 13/8 bit times. This carrier sense information is used to mark the start of the interframe space time and the back off time. The 82588 also defers transmission when the carrier sense is active.

When operating in the NRZI encoded mode, carrier sense is turned off if RxD pin is in the idle state for 8 bit times or more (see Figure 18).

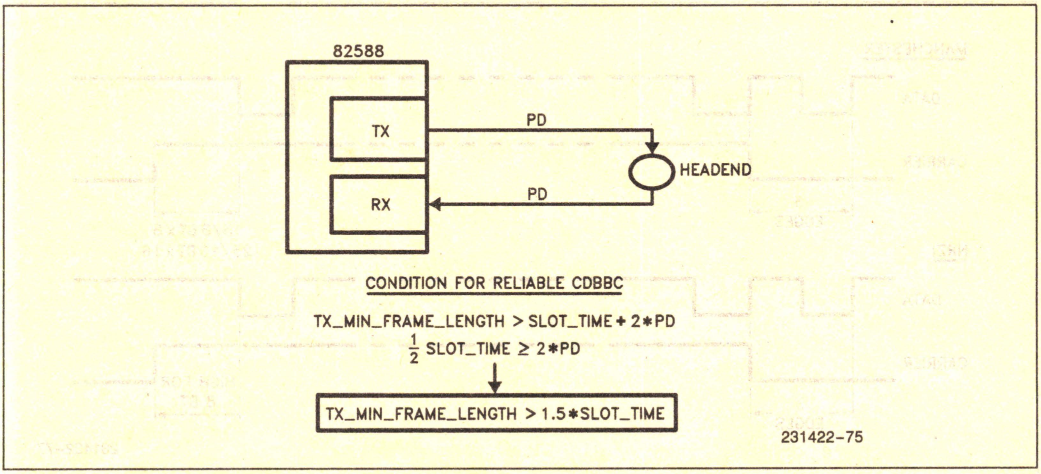


Figure 16. Limitation of CDBBC Mechanism

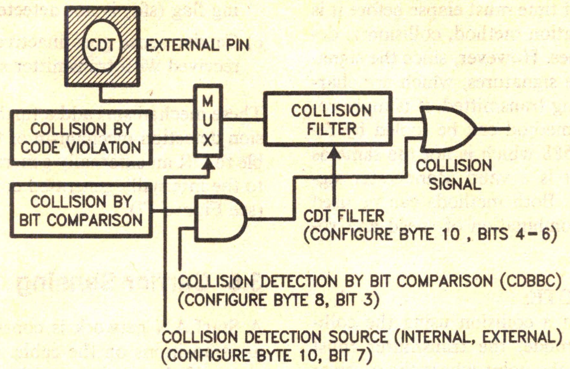


Figure 17. Mode 0, Collision Detection

3.7 Squelching the Input

Squelch circuit is used to filter idle noise on the receiver input. Basically two types of squelch may be used: Voltage and time. Voltage squelch is done to filter out signals whose strength is below a defined voltage threshold (0.6 volts for StarLAN). It prevents idle line noise from disturbing the receive circuits on the controller. The voltage squelch circuit is placed right after the receiving pulse transformer. It enables the input to the RxD pin of the 82588 only when the signal strength is above the threshold.

If the signal received has the proper level but not the proper timing, it should not bother the receiver. This is accomplished by the time squelch circuit on the 82588. Time squelching is essential to weed out spikes, glitches and bad signal especially at the beginning of a frame. The 82588 does not turn on its carrier sense (or receive enable) signal until it receives three consecutive edges, each separated by time periods greater than the fast time clock high time but less than 13/8 bit-times as shown in Figure 18.

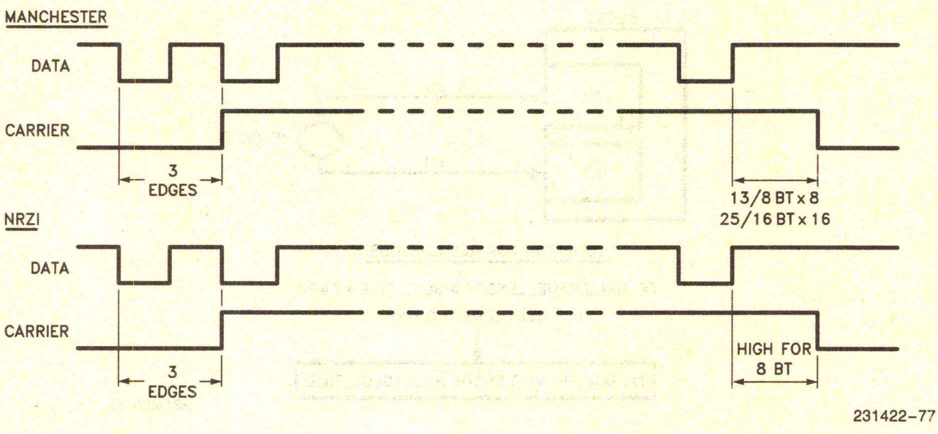


Figure 18. Carrier Sensing

The carrier sense activation can be programmed for a further delay by up to 7 bit times by a configuration parameter called carrier sense filter.

3.8 System Bus Interface

The 82588 has a conventional bus interface making it very easy to interface to any processor bus. Figure 19 shows that it has an 8 bit data bus, read, write, chip select, interrupt and reset pins going to the processor bus. It also needs an external DMA controller for data transfer. A system clock of up to 8 MHz is needed. The read and write access times of the 82588 are very short—95 ns—as shown by Figure 20. This further facilitates interfacing the controller to almost any processor.

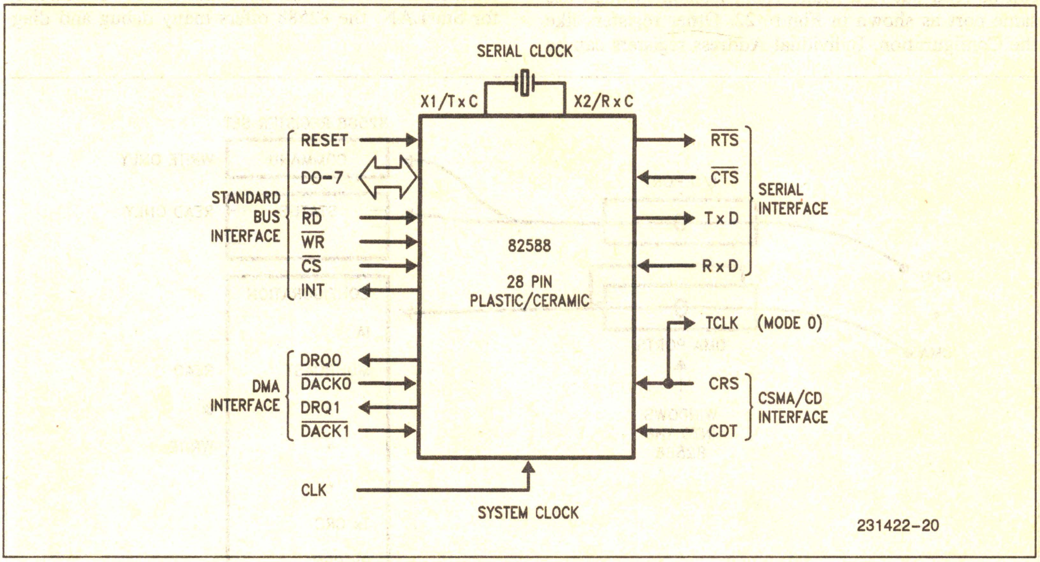


Figure 19. Chip Interface

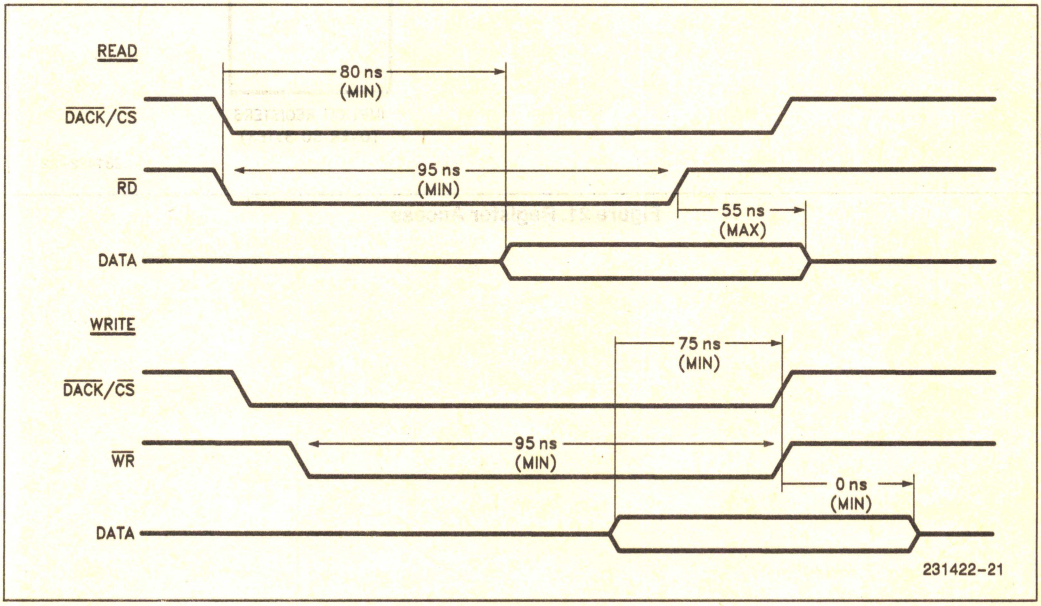


Figure 20. Access Times

The 82588 has over 50 bytes of registers, and most are accessed only indirectly. Figure 21 shows the register access mechanism of the 82588. It has one I/O port and 2 DMA channel ports. These are the windows into the 82588 for the CPU and the DMA controller. An external CPU can write into the Command register and read from the Status registers using I/O instructions and asserting chip select and write or read lines. Although there is just one I/O port and 4 status registers, they can be read out in a round robin fashion through the same port as shown in Figure 22. Other registers like the Configuration, Individual Address registers can be accessed only through DMA. All the internal registers can be dumped into memory by DMA using the Dump command. The execution of some of the commands is described in section 4. See the 82588 Reference Manual for details on these commands.

3.9 Debug and Diagnostic Aids

Besides the standard functions that can be used directly for StarLAN, the 82588 offers many debug and diag-

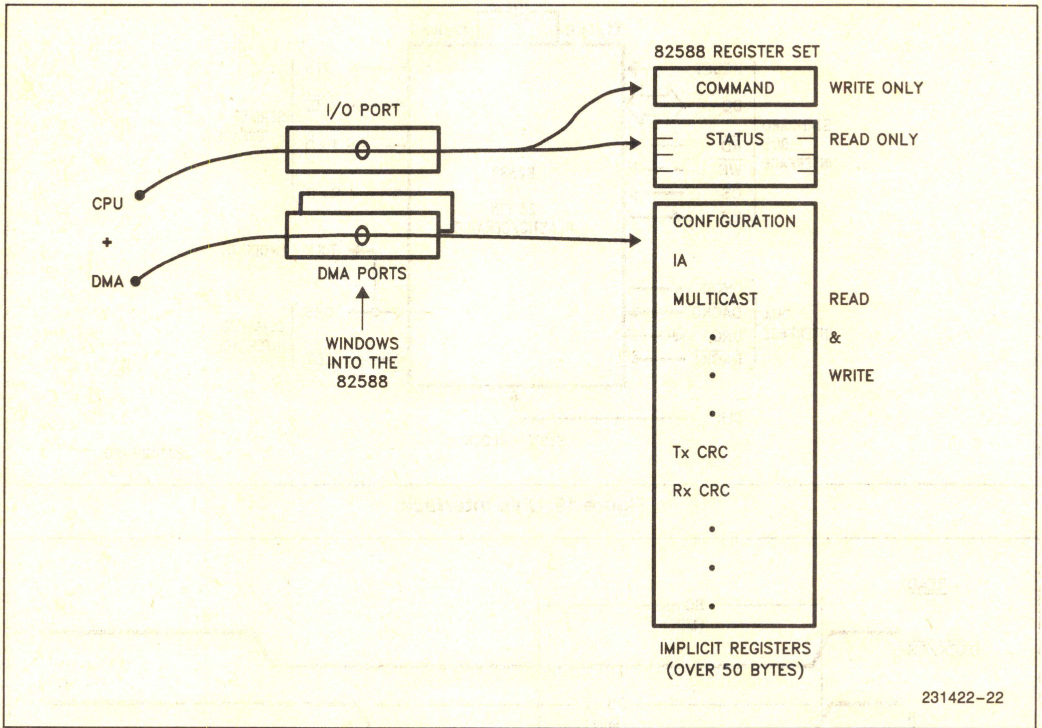
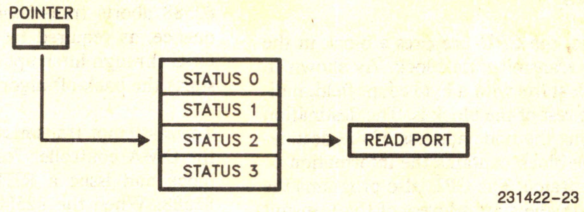


Figure 21. Register Access

4 Status registers are accessed through one read port



The pointer can be changed using a command or can be automatically incremented.

```
READ_STATUS_588: PROCEDURE;                  /* COMMAND 15 */
      OUTPUT (CS_588) = 15;              /* RELEASE POINTER, INITIAL = 00 */
      STATUS_588(0)=INPUT (CS_588);      /* REFRESH STATUS REGISTER IMAGE */
      STATUS_588(1)=INPUT (CS_588);      /* IN MEMORY.
      STATUS_588(2)=INPUT (CS_588);
      STATUS_588(3)=INPUT (CS_588);
      RETURN
END READ_STATUS_588;
```

READING 4 STATUS REGISTERS

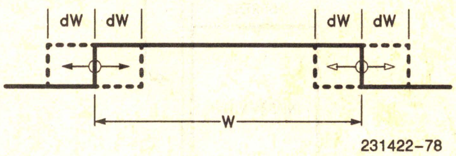
Figure 22. Reading the Status Register

nostics functions. The DIAGNOSE command of the 82588 does a self-test of most of the counters and timers in the 82588 serial unit. Using the DUMP command, all the internal registers of the 82588 can be dumped into the memory. The TDR command does Time Domain Reflectometery on the network. The 82588 has two loopback modes of operation. In the internal loopback mode, the TXD line is internally connected to the RXD one. No data appears outside the chip, and the 82588 is isolated from the link. This mode enables checking of the receive and transmit machines without link interference. In the external loopback mode, the 82588 becomes a full duplex device, being able to receive its own transmitted frames. In this mode data goes through the link and all CSMA/CD mechanisms are involved.

3.10 Jitter Performance

When the 82588 receives a frame from the HUB, the signal has jitter. Jitter is the shifting of the edges of the signal from their nominal position due to the transmission over a length of cable. Many factors like, intersymbol interference (pulses of different widths have different delays through the transmission media), rise and fall times of drivers and receivers, cross talk etc., contribute to the jitter. StarLAN specifies a maximum jitter of ±62.5 ns whenever the signal goes from a NODE/HUB or HUB/HUB. Figure 23 shows that the jitter tolerance of the 82588 is exactly the required ±62.5 ns at 1 Mbs for both 8X, 16X Manchester encoded data.

Jitter = ± variation of an edge from its nominal position.
Jitter can occur on every edge.



	x8	x16
Manchester	±1/16	±1/16
NRZI (Code Violations Enabled)	±1/16 BT	±3/32 BT
NRZI (Code Violations Disabled)	±3/16 BT	±3/16 BT

Figure 23. 82588 Jitter Performance

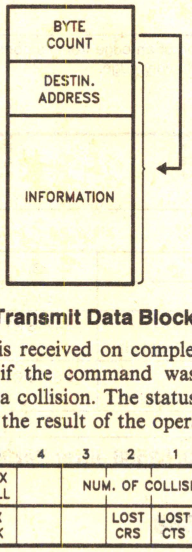
4.0 THE 82588

This chapter describes the basic 82588 operations. Please refer to the 82588 reference manual in Intel Microcommunications Handbook for a detailed description. Basic operations like transmitting a frame, receiving a frame, configuring the 82588 and dumping the register contents are discussed here to give a feel for how the 82588 works.

4.1 Transmit and Retransmit Operations

To transmit a frame, the CPU prepares a block in the memory called the transmit data block. As shown in Figure 24, this block starts with a byte count field, indicating how long the rest of the block is. The destination address field contains the node address of the destination. The rest of the block contains the information or the data field of the frame. The CPU also programs the DMA controller with the start address of the transmit data block. The DMA byte count must be equal to or greater than the block length. The 82588 is then issued a TRANSMIT command—an OUT instruction to the command port of the 82588. The 82588 starts generating DMA requests to read in the transmit data block by DMA. It also determines whether and how long it must defer on the link and after that, it starts transmitting the preamble. The 82588 constructs the frame on the fly. It takes the destination address from the memory, source address from its own individual address memory (previously programmed), data field from the memory and the CRC, is generated on chip, at the end of the frame.

1. Prepare Transmit Data—Block in Memory
2. Program DMA Controller
3. Issue Transmit Command on the Desired Channel



231422-25

Transmit Data Block

4. Interrupt is received on completion of command or if the command was aborted or there was a collision. The status bytes 1 and 2 indicate the result of the operation.

7	6	5	4	3	2	1	0	
TX DEF	HRT BEAT	MAX COLL		NUM. OF COLLISIONS				STATUS 1
COLL		TX OK			LOST CRS	LOST CTS	UNDER RUN	STATUS 2

231422-26

Transmit & Retransmit Results Format

Figure 24. Transmit Operation

At the conclusion of transmission the 82588 generates an interrupt to the CPU. The CPU can then read the status registers to find out if the transmission was successful. If a collision occurs during transmission, the 82588 aborts transmission and generates the jam sequence, as required by IEEE 802.3, and informs the CPU through interrupt and the status registers. It also starts the back-off algorithm.

To re-attempt transmission, the CPU must reinitialize the DMA controller 7to the start of the transmit data block and issue a RETRANSMIT command to the 82588. When the 82588 receives the retransmit command and the back-off timer has expired, it transmits again. Interrupt and the status register contents again indicate the success or failure of the (re)transmit attempt.

The main difference between transmit and retransmit commands is that retransmit does not clear the internal count for the number of collisions occurred, whereas transmit does. Moreoever, retransmit takes effect only when the back-off timer has expired.

4.2 Configuring the 82588

To initialize the 82588 and program its network and system parameters, a configure operation is performed. It is very similar to the transmit operation. Instead of a transmit data block as in transmit command, a configure data block—shown in Figure 12—is prepared by the CPU in the memory. The first two bytes of the block specify the length of the rest of the block, which specify the network and system parameters for the 82588. The DMA controller is then programmed by the CPU to the beginning of this block and a CONFIGURE command is issued to the 82588. The 82588 reads in the parameters by DMA and loads the parameters in the on-chip registers.

Similarly, for programming the INDIVIDUAL ADDRESS and MULTICAST ADDRESSes, the DMA controller is used to load the 82588 registers.

4.3 Frame Reception

Before enabling the 82588 for reception the CPU must make a buffer available for the frame to be received. The CPU must program the DMA controller with the starting address of the buffer and then issue the RX_ENABLE command to the 82588. When a frame arrives at the RxD pin of the 82588, it starts being received. Only if the address in the destination address matches either the Individual address, Multicast address or if it is a broadcast address, is the frame deposited into memory by the 82588 using DMA. The format of storage in the memory is shown in Figure 25. At the end, a two byte field is attached which shows the status of the received frame. If CRC, alignment or overrun errors are encountered, they are reported. An inter-

1. Prepare a Buffer for Reception
2. Program DMA Controller
3. Issue Receiver Enable Command

When a frame is received, it is deposited in the memory. Receive status bytes (2) are appended to the frame in the memory, byte count written in the status registers 1, 2, and an interrupt is generated.

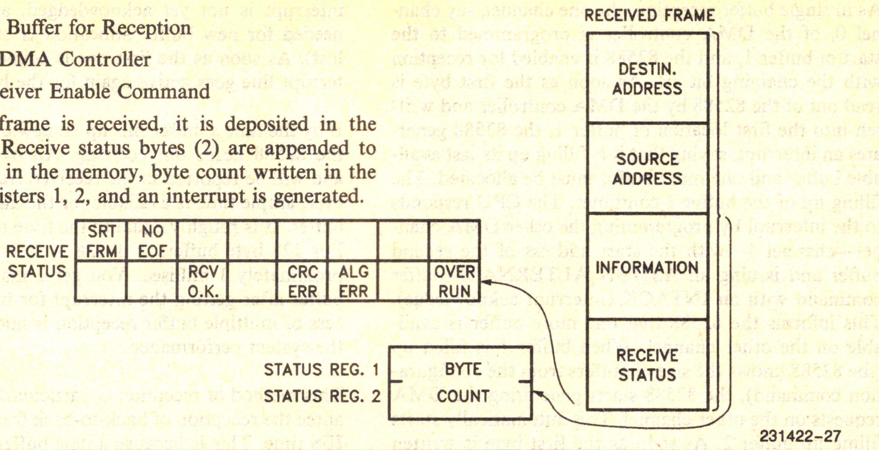


Figure 25. Receive Operation (Single Buffer)

rupt from 82588 occurs when all the bytes have been transferred to the memory. This informs the CPU that a new frame has been received.

If the received frame has errors, the CPU must recover (or re-use) the buffer. Note that the entire frame is deposited into one buffer. The 82588 when NOT configured for the external loopback mode, will detect collisions (code violations) during receptions. If a collision is detected, the reception is aborted and status updated. CPU is then informed by an interrupt (if the collided frame fragment is shorter than the address length, no reception will be started), and no interrupt will happen.

4.3.1 Multiple Buffer Frame Reception

It is also possible to receive a frame into a number of fixed size buffers. This is particularly economical if the received frames vary widely in size. If the single buffer scheme were used as described above, the buffer required would have to be bigger than the longest expected frame and would be very wasteful for very short (typically acknowledge or control) frames. The multiple buffer reception is illustrated in Figure 26. It uses two DMA channels for reception.

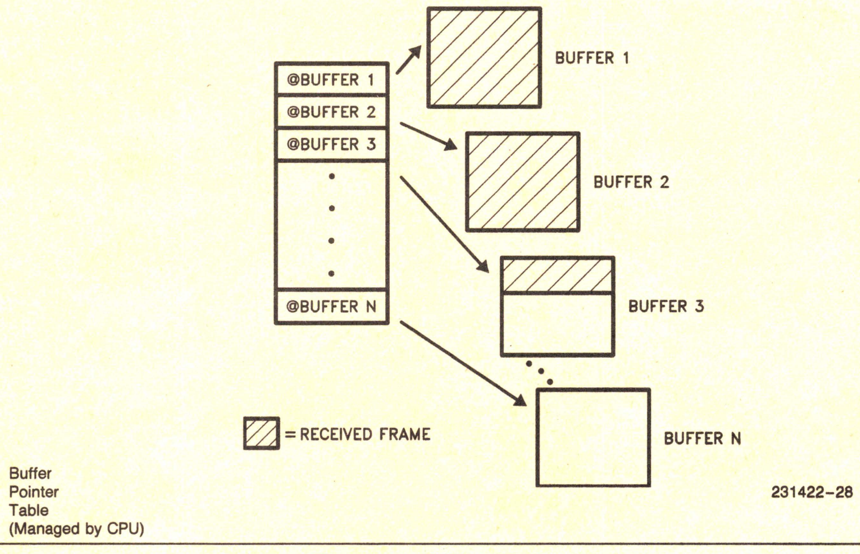


Figure 26. Multiple Buffer Reception

As in single buffer reception, the one channel, say channel 0, of the DMA controller is programmed to the start of buffer 1, and the 82588 is enabled for reception with the chaining bit set. As soon as the first byte is read out of the 82588 by the DMA controller and written into the first location of buffer 1, the 82588 generates an interrupt, saying that it is filling up its last available buffer and one more buffer must be allocated. The filling up of the buffer 1 continues. The CPU responds to the interrupt by programming the other DMA channel—channel 1—with the start address of the second buffer and issuing an ASSIGN ALTERNATE buffer command with an INTACK (interrupt acknowledge). This informs the 82588 that one more buffer is available on the other channel. When buffer 1 is filled up (the 82588 knows the size of buffers from the configuration command), the 82588 starts generating the DMA requests on the other channel. This automatically starts filling up buffer 2. As soon as the first byte is written into buffer 2, the 82588 interrupts the CPU again asking for one more buffer. The CPU programs the channel 0 of the DMA controller with the start address of buffer 3, issues an ASSIGN ALTERNATE buffer command with INTACK. This keeps the buffer 3 ready for the 82588. This switching of channels continues until the entire frame is received generating an end of frame interrupt. The CPU maintains the list of pointers to the buffers used.

Since a new buffer is allocated at the time of filling up of the last buffer, the 82588 automatically switches to the new buffer to receive the next frame as soon as the last frame is completely received. It can start receiving the new frame almost immediately, even before the end of frame interrupt is serviced and acknowledged by the CPU. If a new frame comes in, and the previous frame interrupt is not yet acknowledged, another interrupt needed for new buffer allocation is buffered (and not lost). As soon as the first one is acknowledged, the interrupt line goes active again for the buffered one.

If by the time a buffer fills up no new buffer is available, the 82588 keeps on receiving. An overrun will occur and will be reported in the received frame status. However, ample time is available for the allocation of a new buffer. It is roughly equal to the time to fill up a buffer. For 128 byte buffers it is 128 × 8 = 1024 ms or approximately 1 millisec. You get 1 ms to assign a new buffer after getting the interrupt for it. Hence the process of multiple buffer reception is not time critical for the system performance.

This method of reception is particularly useful to guarantee the reception of back-to-back frames separated by IFS time. This is because a new buffer is always available for the new frame after the current frame is received.

Although both the DMA channels get used up in receiving, only one channel is kept ready for reception and the other one can be used for other commands until the reception starts. If an execution command like transmit or dump command is being executed on a channel which must be allocated for reception, the command gets automatically aborted when the ASSIGN ALTERNATE BUFFER command is issued to the channel used for the execution command. The interrupt for command abortion occurs after the end of frame interrupt.

4.4 Memory Dump of Registers

All the 82588 internal registers can be dumped in the memory by the DUMP command. A DMA channel is used to transfer the register contents to the memory. It is very similar to reception of a frame; instead of data from the serial link, the data from the registers gets written into the memory. This provides a software debugging and diagnostic tool.

4.5 Other Operations

Other 82588 operations like DIAGNOSE, TDR, ABORT, etc. do not require any parameter or data transfer. They are executed by writing a command to the 82588 command register and knowing the results (if any) through the status registers.

5.0 StarLAN NODE FOR IBM PC

This chapter deals with the hardware—the StarLAN board—to interface the IBM PC to a StarLAN Network. This is a slave board which takes up one slot on the I/O channel of the IBM PC. Figure 27 shows an abstract block diagram of the board. It requires the IBM PC resources of the CPU, memory, DMA and interrupt controller on the system board to run it. Such a board has two interfaces. The IBM PC I/O Channel on the system or the parallel side and the telephone grade twisted pair wire on the serial side. Figures 28, 29 show the circuit diagram of the board.

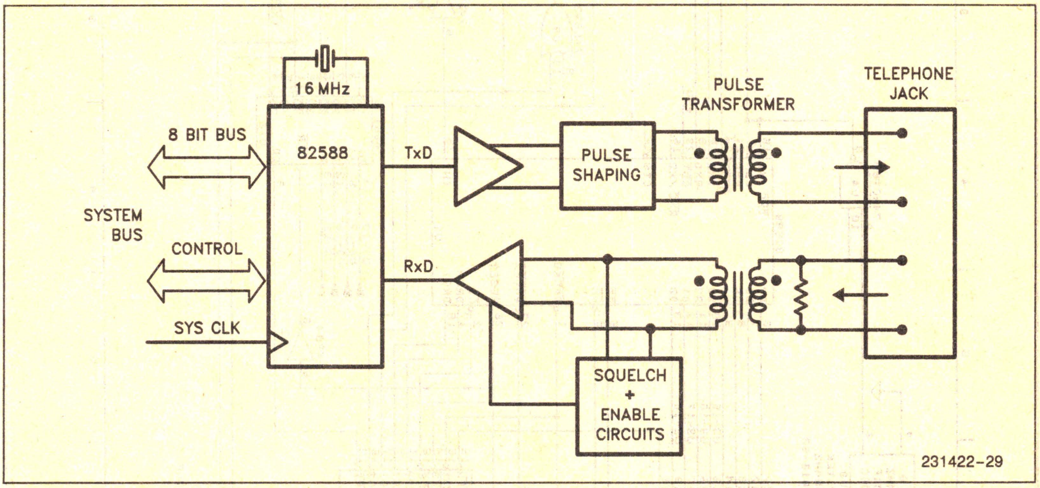


Figure 27. 82588 Based StarLAN Node

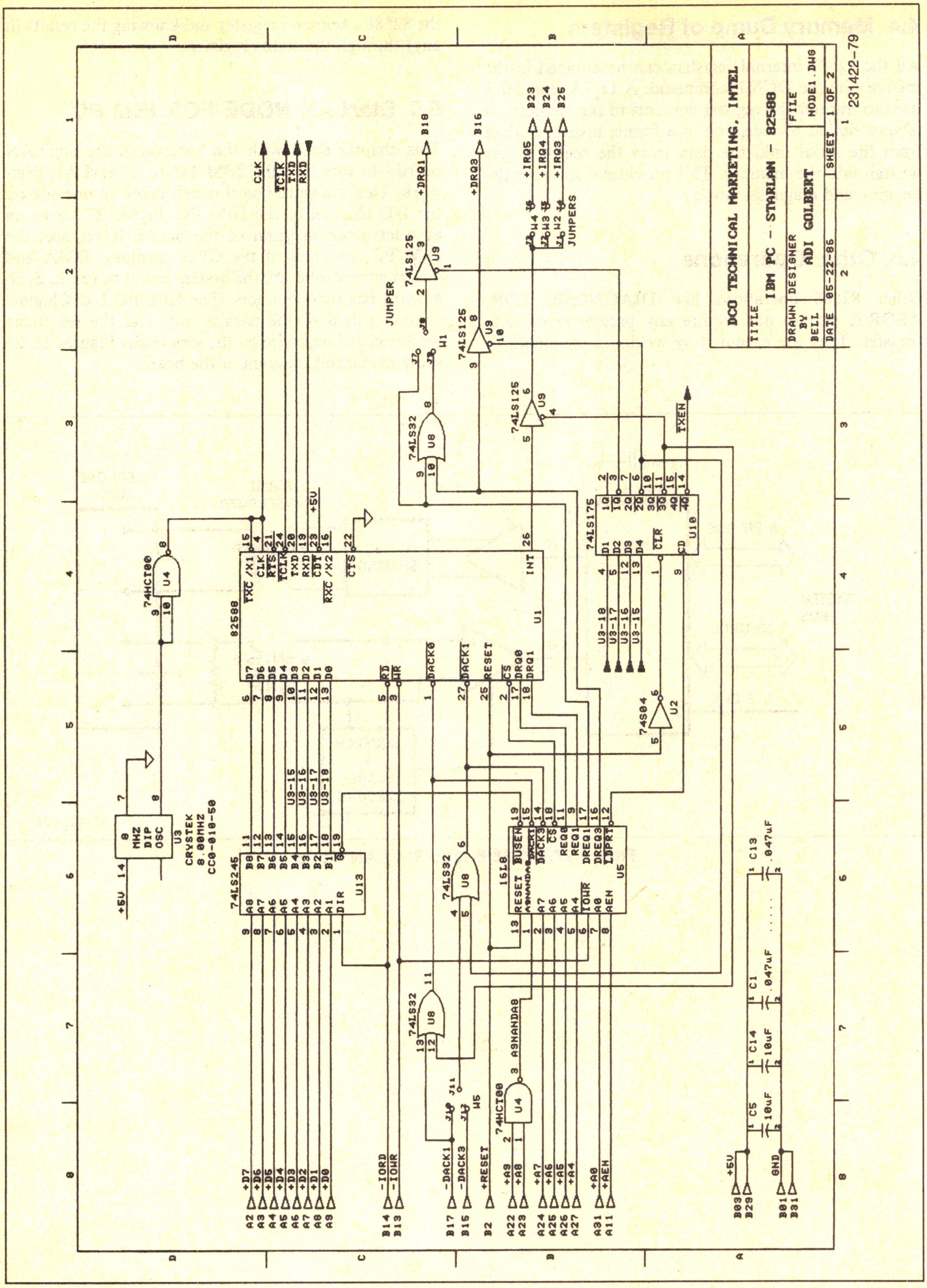

DCO TECHNICAL MARKETING, INTEL
TITLE
IBM PC - STARLAN - 82588
DRAWN BY
BELL
DESIGNER
ADI GOLBERT
FILE
NODE1.DWG
DATE
05-22-86
SHEET
1 OF 2
231422-79
82588
U1
74LS245
U13
74HCT00
U4
74LS32
U8
74LS125
U9
74LS175
U10
74S04
U2
16L8
U5
8 MHZ DIP OSC
U3
CRYSTEK
8.00MHZ
CCO-010-50
JUMPER
JUMPERS
CLK
TCLK
TXD
RXD
TXEN
+DRQ1
+DRQ3
+IRQ5
+IRQ4
+IRQ3
+5V
GND
C13
.047uF
C1
.047uF
C14
10uF
C5
10uF

Figure 28

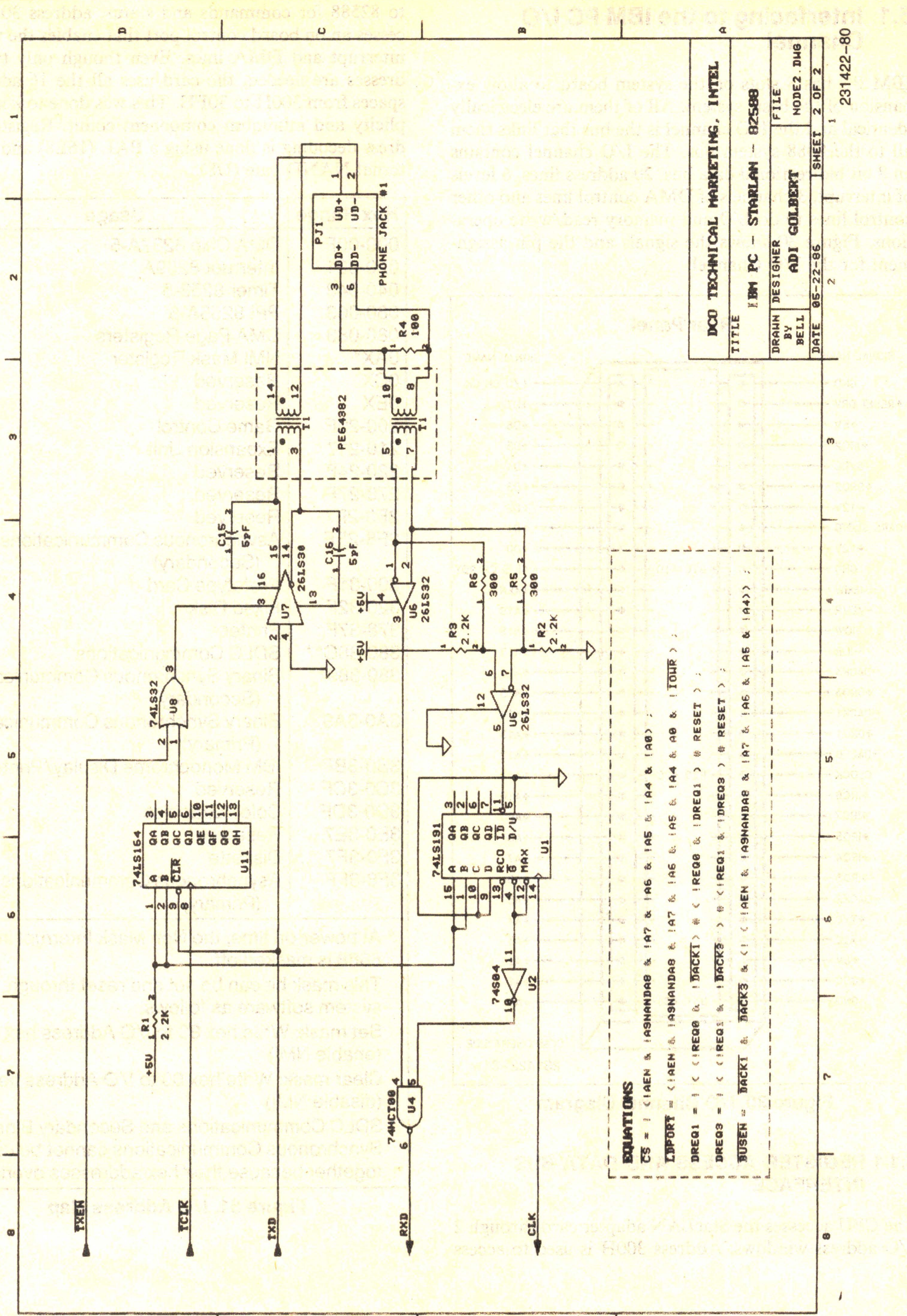

DCD TECHNICAL MARKETING, INTEL
TITLE IBM PC - STARLAN - 82588
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE NODE2.DWG
DATE 05-22-86
SHEET 2 OF 2
231422-80
PJ1
PHONE JACK #1
PE64382
26LS30
26LS32
74LS32
74LS164
74LS191
74S04
74HCT00
EQUATIONS
CS = ! (!AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & !A0) ;
LDPORT = (!AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & A0 & !IOWR) ;
DREQ1 = ! ((!REQ0 & !DACK1) # (!REQ0 & !DREQ1) # RESET) ;
DREQ3 = ! ((!REQ1 & !DACK3) # (!REQ1 & !DREQ3) # RESET) ;
BUSEN = DACK1 & DACK3 & (! (!AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4)) ;
TXEN
TCLK
TXD
RXD
CLK

Figure 29

5.1 Interfacing to the IBM PC I/O Channel

IBM PC has 8 slots on the system board to allow expansion of the basic system. All of them are electrically identical and the I/O channel is the bus that links them all to the 8088 system bus. The I/O channel contains an 8 bit bidirectional data bus, 20 address lines, 6 levels of interrupt, 3 channels of DMA control lines and other control lines to do I/O and memory read/write operations. Figure 30 shows the signals and the pin assignment for the I/O Channel.

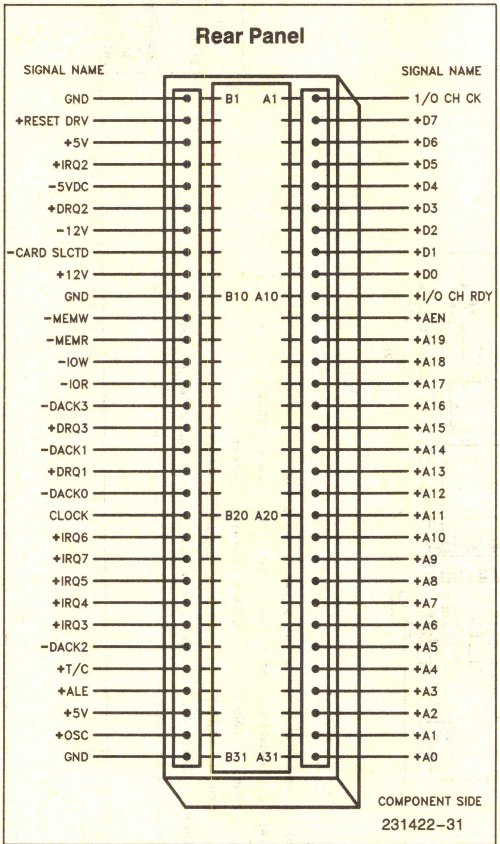


Figure 30. I/O Channel Diagram

5.1.1 REGISTER ACCESS AND DATA BUS INTERFACE

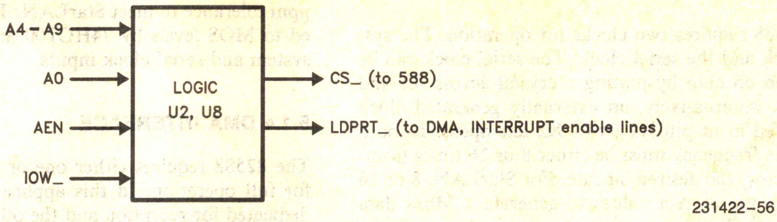
The CPU accesses the StarLAN adapter card through 2 I/O address windows. Address 300H is used to access to 82588 for commands and status, address 301H accesses an on board control port that enables the various interrupt and DMA lines. Even though only two addresses are needed, the card uses all the 16 addresses spaces from 300H to 30FH. This was done to keep simplicity and minimum component count. Registers address decoding is done using a PAL (16L8) and an external NAND gate (U8).

Hex Range	Usage
000-00F	DMA Chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Registers
0AX*	NMI Mask Register
0CX	Reserved
0EX	Reserved
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C**	SDLC Communications
380-389**	Binary Synchronous Communications (Secondary)
3A0-3A9	Binary Synchronous Communications (Primary)
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

* At power-on time, the Non Mask Interrupt into the 8088 is masked off.
This mask bit can be set and reset through system software as follows:
Set mask: Write hex 80 to I/O Address hex A0 (enable NMI)
Clear mask: Write hex 00 to I/O Address hex A0 (disable NMI)

** SDLC Communications and Secondary Binary Synchronous Communications cannot be used together because their hex addresses overlap.

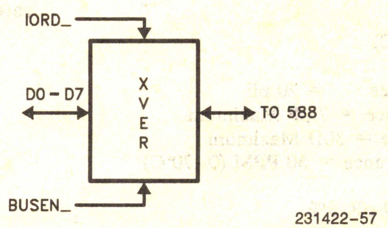
Figure 31. I/O Address Map



Register Access

```
Format of Following Equations Will Be According To

The Following Specifications:

    ! INVERT

    _ SIGNAL ACTIVE LOW

    & LOGIC AND

    # LOGIC OR

A9NANDA8 = ! (A9 & A8)

CS_ = ! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & !A0 )

LDPORT_ = ! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4 & A0 & !IOWR_ )

BUSEN_ = DACK1_ & DACK2_ & (! ( !AEN & !A9NANDA8 & !A7 & !A6 & !A5 & !A4));
```

The signal CS__ decodes address 300H, it is only active when AEN is inactive meaning CPU and not DMA cycles. LDPORT__ has exactly the same logic for address 301H, but it is only active during I/O write cycles. The I/O port sitting on address 301H is write only. The data BUS lines D0 to D7 are buffered from the 82588 to the PC bus using an 74LS245 transceiver chip.



Data Bus Interface

The Bus transceiver is enabled if: A DMA access is taking place, or I/O ports 300H to 30FH are being accessed.

5.1.2 Control Port

As mentioned the StarLAN adapter port has a 4-bit write only control port. The purpose of this port is to selectively enable the DMA and INTERRUPT request lines. Also it can completely disable the transmitter.

Control Port Definition

ENDRQ1	ENDRQ3	ENINTER	TXEN

ENDRQ1, ENDRQ2	: "1" Enable DMA requests.
ENINTER	: "1" Enable INTERRUPT request.
TXEN	: "1" Enable the transmitter.

On power up all bits default to "0".

5.1.3 CLOCK GENERATION

The 82588 requires two clocks for operation. The system clock and the serial clock. The serial clock can be generated on chip by putting a crystal across X1 and X2 pins. Alternatively, an externally generated clock can be fed in at pin X1 (with X2 left open). In both cases, the frequency must be either 8 or 16 times (sampling factor) the desired bit rate. For StarLAN, 8 or 16 MHz are the correct values to generate 1 Mb/s data rate. A configuration parameter is used to tell the 82588 what the sampling factor is. An externally supplied clock must have MOS levels (0.6V–3.9V). Specifications for the crystal and the circuit are shown in Figure 32.

The system clock has to be supplied externally. It can be up to 8 MHz. This clock runs the parallel side of the 82588. Its frequency does not have any impact on the read and write access times but on the rate at which data can be transferred to and from the 82588 (Maximum DMA data rate is one byte every two system clocks). This clock doesn't require MOS levels.

The I/O channel of the IBM PC supplies a 4.77 MHz signal of 33% duty cycle. This signal could be used as a system clock. It was decided, however, to generate a separate clock on the StarLAN board to be independent of the I/O channel clock so that this board can also be used in other IBM PCs and also in some other compatibles. The 8 MHz system clock is generated using a DIP OSCILLATOR which have the required 50 ppm tolerance to meet StarLAN. This clock is converted to MOS levels by 74HCT00 and fed into both the system and serial clock inputs.

5.1.4 DMA INTERFACE

The 82588 requires either one or two DMA channels for full operation. In this application, one channel is dedicated for reception and the other is used for transmissions and the other commands. Use of only one DMA channel is possible but may require more complex software, also some RX frames may be lost during switches of the DMA channel from the receiver to the transmitter (Those frames will be recovered by higher layers of the protocol). Also using only one DMA channel will limit the 82588 loopback functionality. So the recommendation is to operate with two DMA channels if available. Appendix C describes a method of operating with only one DMA channel without loosing RX frames.

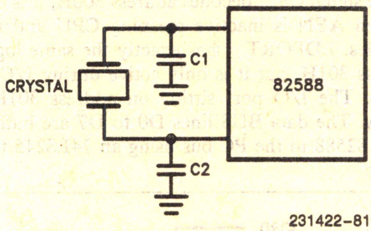
The IBM PC system board has one 8237A DMA controller. Channel 0 is used for doing the refresh of DRAMs. Channels 1, 2 and 3 are available for add-on boards on the I/O Channel. The floppy disk controller board uses the DMA channel 2 leaving exactly two channels (1 and 3) for the 82588. The situation is worse if the IBM PC/XT is used, since it uses channel 3 for the Winchester hard disk leaving just the channel 1 for

Series Resonance

—Frequency Will Drift by About 400 PPM from Nominal
—No Capacitors Needed
—Doesn't Meet StarLAN Requirements

Meeting StarLAN 100 PPM Requirements

—Use Parallel Resonance Crystal
—Recommended For Precise Frequencies
—82588 X-TAL Oscillator Stability ±35 PPM (0–70°C)



Crystal: Load Capacitance = 20 pF
Shunt Capacitance = 7 pF Maximum
Series Resistance = 30Ω Maximum
Frequency Tolerance = 50 PPM (0–70°C)

C1, C2 → 27 pF or 39 pF, 5%

Figure 32. Crystal Specifications

the 82588. On the other hand, the IBM PC/AT has 5 free DMA channels. We will assume that 8237A DMA channels 1 and 3 are available for the 82588 as in the case of the IBM PC.

Since the channel 0 of 8237A is used to do refresh of DRAMs all the channels should be operated in single byte transfer mode. In this mode, after every transfer for any channel the bus is granted to the current highest priority channel. In this way, no channel can hog the bus bandwidth and, more important, the refresh of DRAMs is assured every 15 microseconds since the refresh channel (number 0) has the highest priority. This mode of operation is very slow since the HOLD is dropped by the 8237A and then asserted again after every transfer. Demand mode of operation is a lot more suitable to 82588 but it cannot be used because of the refresh requirements.

Whenever the 82588 interfaces to the 8237A in the single transfer mode, there is a potential 8237A lock-up problem. The 82588 may deactivate its DMA request line (DREQ) before receiving an acknowledge from the DMA controller. This situation may happen during command abortions, or aborted receptions. The 8237A under those circumstances may lock-up. In order to solve this potential problem, an external logic must be used to insure that DREQ to the DMA controller is never deactivated before the acknowledge is received. Figure 33 shows the logic to implement this function. This logic is implemented in the 16L8 PAL.

The 82588 DREQ lines are connected to the IBM/PC bus through tri-state buffers which are enabled by writing to I/O port 301H. This function enables the use of either one or two DMA channels and also the sharing of DMA channels with other adapter boards.

5.1.5 INTERRUPT CONTROLLER

The 82588 interrupts the CPU after the execution of a command or on reception of a frame. It uses the 8259A interrupt controller on the system board to interrupt the CPU. There are 6 interrupt request lines, IRQ2 to IRQ7, on the I/O channel. Figure 34 shows the assignment of the lines. In fact, none of the lines are completely free for use. To add any new peripheral which uses a system board interrupt, this interrupt needs to have the capability to share the specific line, by driving the line with a tri-state driver. The 82588 StarLAN adapter board can optionally drive interrupt lines IRQ3, IRQ4 or IRQ5 (An 74LS125 driver is used).

Number	Usage
NMI	Parity
0	Timer
1	Keyboard
2	Reserved
3	Asynchronous Communications (Secondary)
	SDLC Communications
	BSC (Secondary)
4	Asynchronous Communications (Primary)
	SDLC Communications
	BSC (Primary)
5	Fixed Disk
6	Diskette
7	Printer

Figure 34. IBM PC Hardware Interrupt Listing

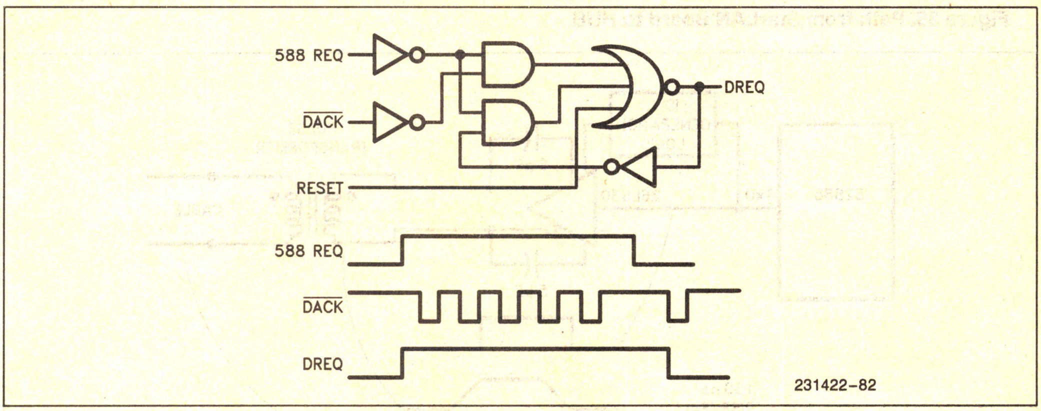


Figure 33. DMA Request Logic

5.2 Serial Link Interface

A typical StarLAN adapter board is connected to the twisted pair wiring using an extension cable (typically up to 8 meters—25 ft.). See Figure 35. One end of the cable plugs into the telephone modular jack on the StarLAN board and the other end into a modular jack in the wall. The twisted pair wiring starts at the modular jack in the wall and goes to the wiring closet. In the wiring closet, another telephone extension cable is used to connect to a StarLAN HUB. The transmitted signal from the 82588 reach the on-board telephone jack through a RS-422 driver with pulse shaping and a pulse transformer. The received signals from the telephone jack to the 82588 come through a pulse transformer, squelch circuit and a receive enable circuit.

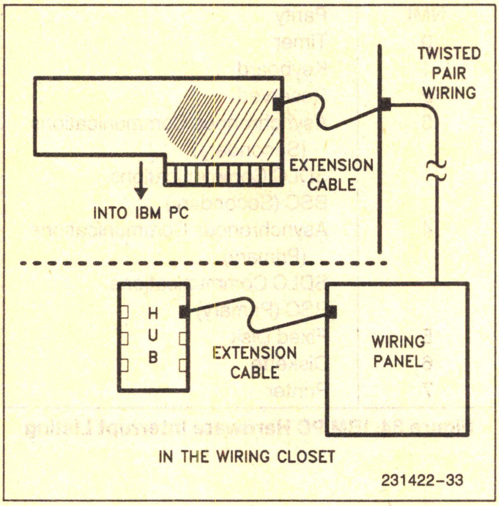


Figure 35. Path from StarLAN Board to HUB

5.2.1 TRANSMIT PATH

The single ended transmit signal on the TxD pin is converted to a differential signal and the rise and fall times are increased to 150 to 200 ns before feeding it to the pulse transformer (this pulse shaping is not a requirement, but proves to give good results). Am26LS30 is a RS-422 driver which converts the TxD signal to a differential signal. It also has slew rate control pins to increase to rise and fall times. A large rise and fall time reduces the possibility of crosstalk, interference and radiation. By the other hand a slower edge rate increases the jitter. In the StarLAN adapter card, the first approach was used. The 26LS30 converts a square pulse to a trapezoidal one—see Figure 36. The filtering effect of the cable further adds to reduce the higher frequency components from the waveform so that on the cable the signal is almost sinusoidal. The pulse transformer is for DC isolation. The pulse transformers from Pulse Engineering—type PE 64382—was used in this design. This is a dual transformer package which introduces an additional rise and fall time of about 70–100 ns on the signal, helping the former discussed waveshaping.

5.2.2 IDLE PATTERN GENERATION

StarLAN requires transmitters to generate an IDLE pattern after the last transmitted data bit. The IDLE pattern is defined to be a constant high level for 2–3 microseconds. The purpose of this pattern is to insure that receivers will decode properly the last transmitted data bits before signal decay. Currently the 82588 needs one external component to generate the IDLE. The operation principle is to have an external shift register (74LS164) that will kind of act as an envelope detector of the TXD line. Whenever the TXD line goes low

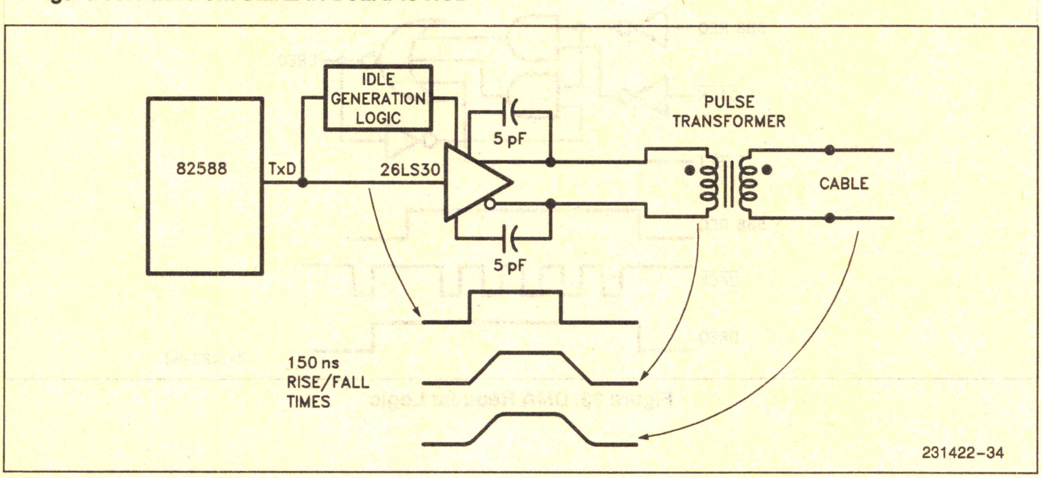


Figure 36. Wave Shaping

(first preamble bit), the output of the shift register (third cell) will immediately go low, enabling the RS-422 driver, the shift register being clocked by TCLK— will time the duration of the TXD high times. If the high time is more than 2 microseconds, meaning that the 82588 has gone idle, the transmitter will be disabled (See Figure 37). Another piece of this logic is the OR-ing of the output of the shift register with TXEN—signal which comes from the board control port. This signal completely disables the transmitter. The other purpose of this enable signal, is to make sure that after power-up, before the 82588 is configured, the RS-422 drivers won't be enabled (TCLK__ is not active before the configure command). See Figures 28, 29 for the complete circuit.

5.3 RECEIVE PATH

The signal coming from the HUB over the twisted pair wire is received on the StarLAN board through a 100Ω line termination resistor and a pulse transformer. The pulse transformer is of the same type as for the transmit side and its function is dc isolation. The received signal which is differential and almost sinusoidal is fed to the Am26LS32 RS-422 receiver. As seen from Figure 38 the pulse transformer feeds two RS-422 receivers. The one on the bottom is for squelch filtering and the one above is the real receiver which does real zero crossing detection on the signal and regenerates a square digital waveform from the sinusoidal signal that is received. Proper zero crossing detection is very essential; if the edges of the regenerated signal are not at zero crossings, the resulting signal may not be a proper Manchester encoded signal (self introduced jitter) even if the original signal is valid Manchester. The resistors in the lower receiver keep its differential inputs at a voltage difference of 600 mV. These bias resistors ensure that the output remains high as long as the input signal is more than −600 mV. It is very important that the RxD pin remains HIGH (not LOW or floating) whenever the receive line is idle. A violation of this may cause the 82588 to lock-up on transmitting. Remember, that based on the signal on the RxD pin, the 82588 extracts information on the data being received, Carrier Sense and Collision Detect. This squelch of 600 mV keeps the idle line noise from getting to the 82588. Figure 39 shows that when the differential input of the receiver crosses zero, a transition occurs at the output. It also shows that if the signal strength is higher than −600 mV, the output does not change. (This kind of squelching is called negative squelching, and it is done due to the fact that the preamble pattern starts with a going low transition). Note that the differential voltage at the upper receiver input is zero when the line is idle. The output of the squelch goes to a pulse stretcher which generates an envelope of the received frame. The envelope is a receive enable signal and is used to AND the signal from the real zero crossing receiver before feeding it to the RxD pin of the 82588.

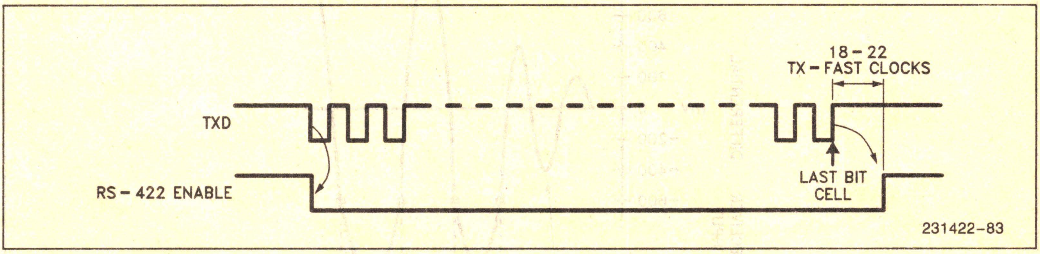


Figure 37. Idle Generation

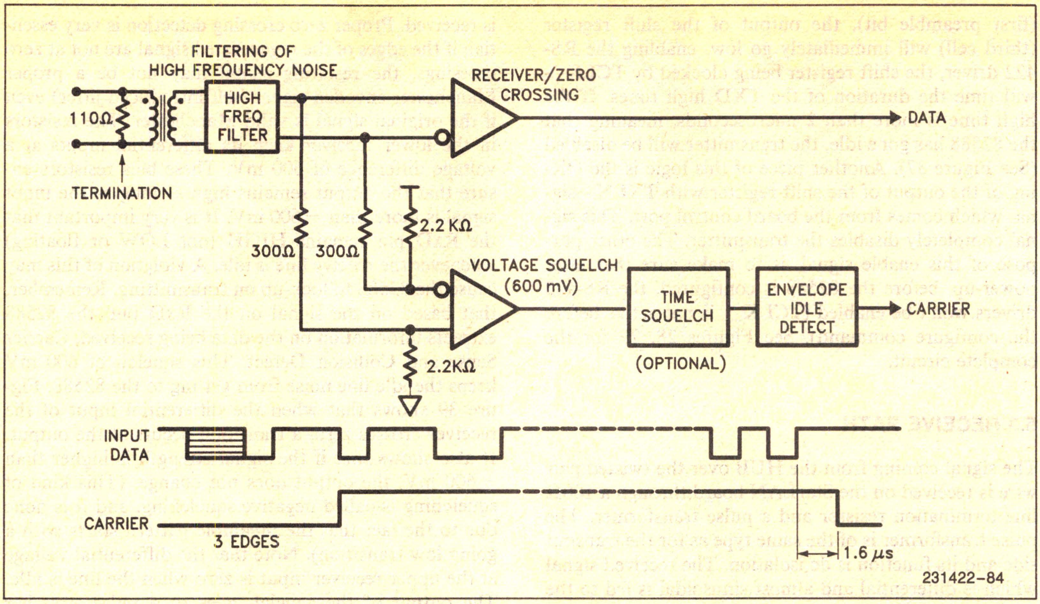


Figure 38. Input Ports

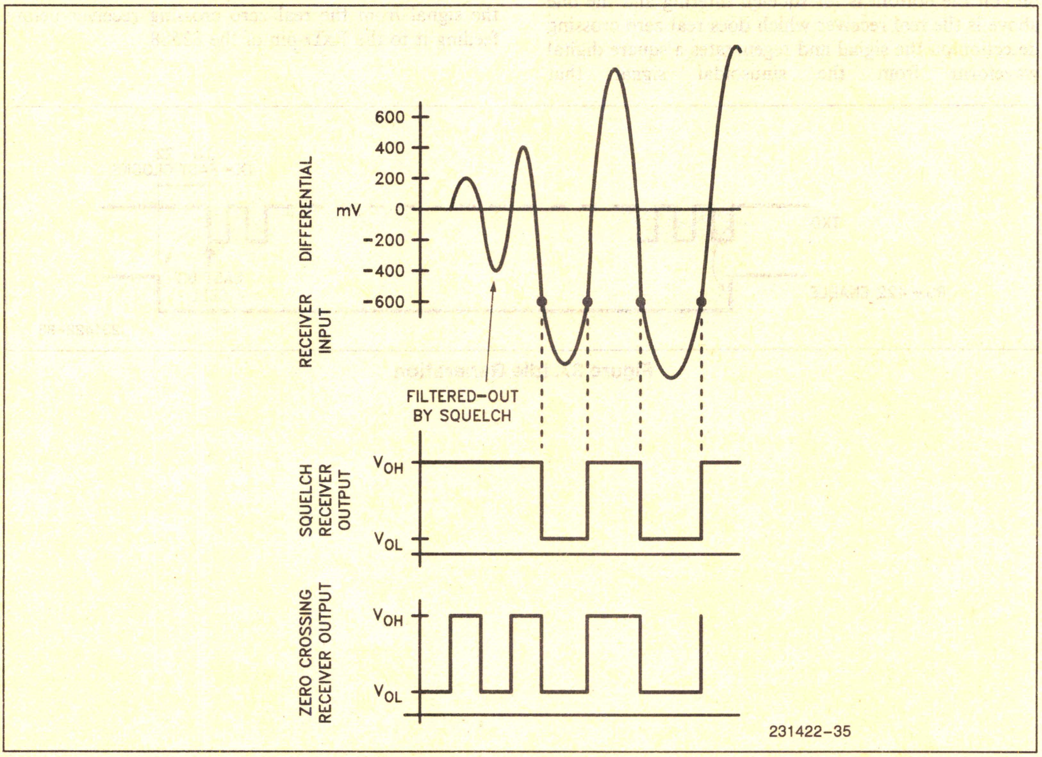


Figure 39. Squelch Circuit Output

5.4 80188 Interface to 82588

Although the 82588 interfaces easily to almost any processor, no processor offers as much of the needed functionality as the 80186 or its 8 bit cousin, the 80188. The 80188 is 8088 object code compatible processor with DMA, timers, interrupt controller, chip select logic, wait state generator, ready logic and clock generator functions on chip. Figure 40 shows how the 82588, in a StarLAN environment interfaces to the 80188. It uses the clock, chip select logic, DMA channels, interrupt controller directly from the 80188. The interface components between the CPU and the 82588 are totally eliminated.

5.5 iSBX Interface to StarLAN

Figure 41 shows how to interface the 82588 in a StarLAN environment to the iSBX bus. It uses 2 DMA channels—tapping the second DMA channel from a neighboring iSBX connector. Such a board can be used to make a StarLAN to an Ethernet or a SNA or DECNET gateway when it is placed on an appropriate SBC board. It may also be used to give a StarLAN access to any SBC board (with an iSBX connector) independent of the type of processor on the board.

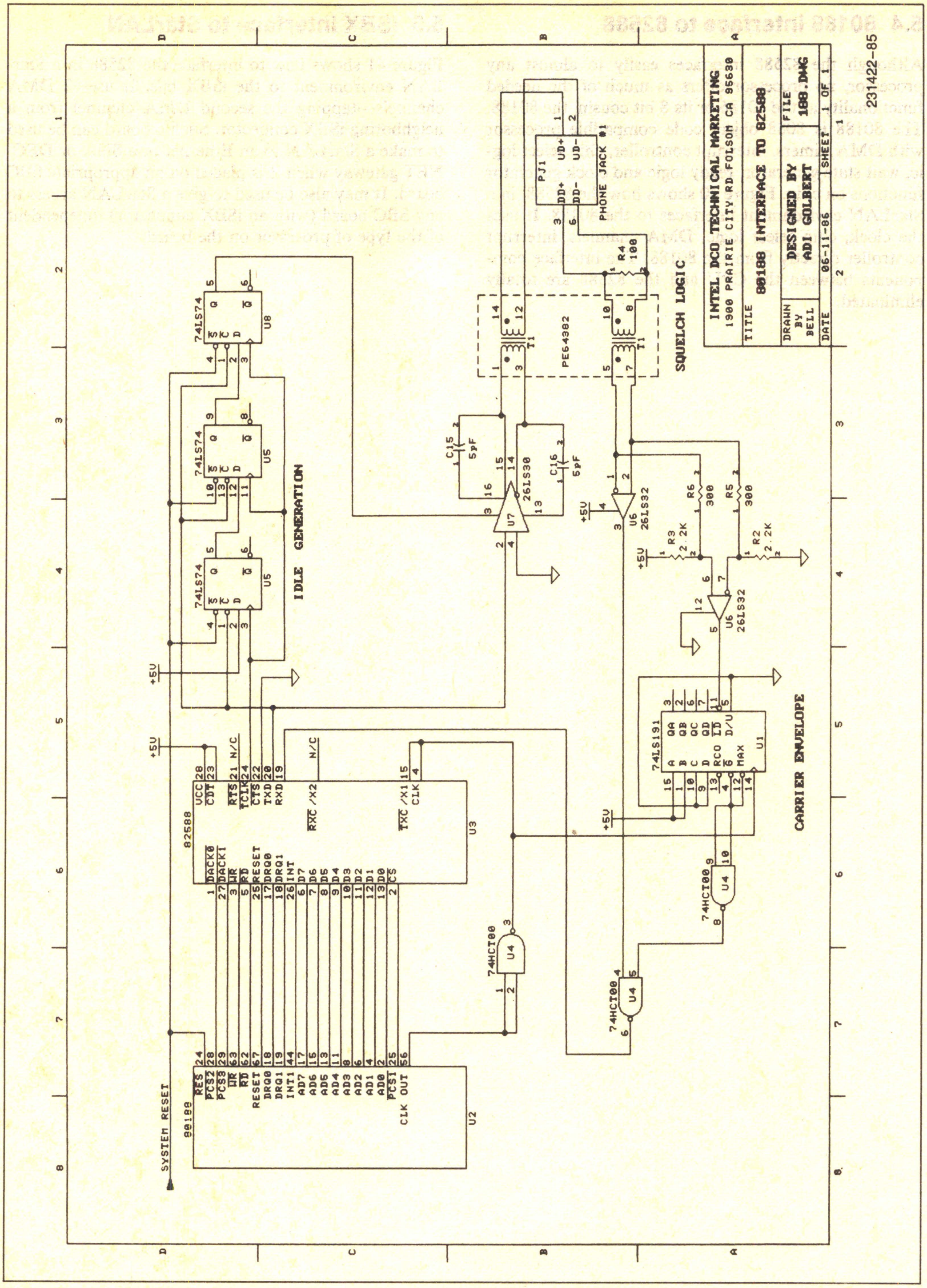


Figure 40. 80188 Interface to 82588

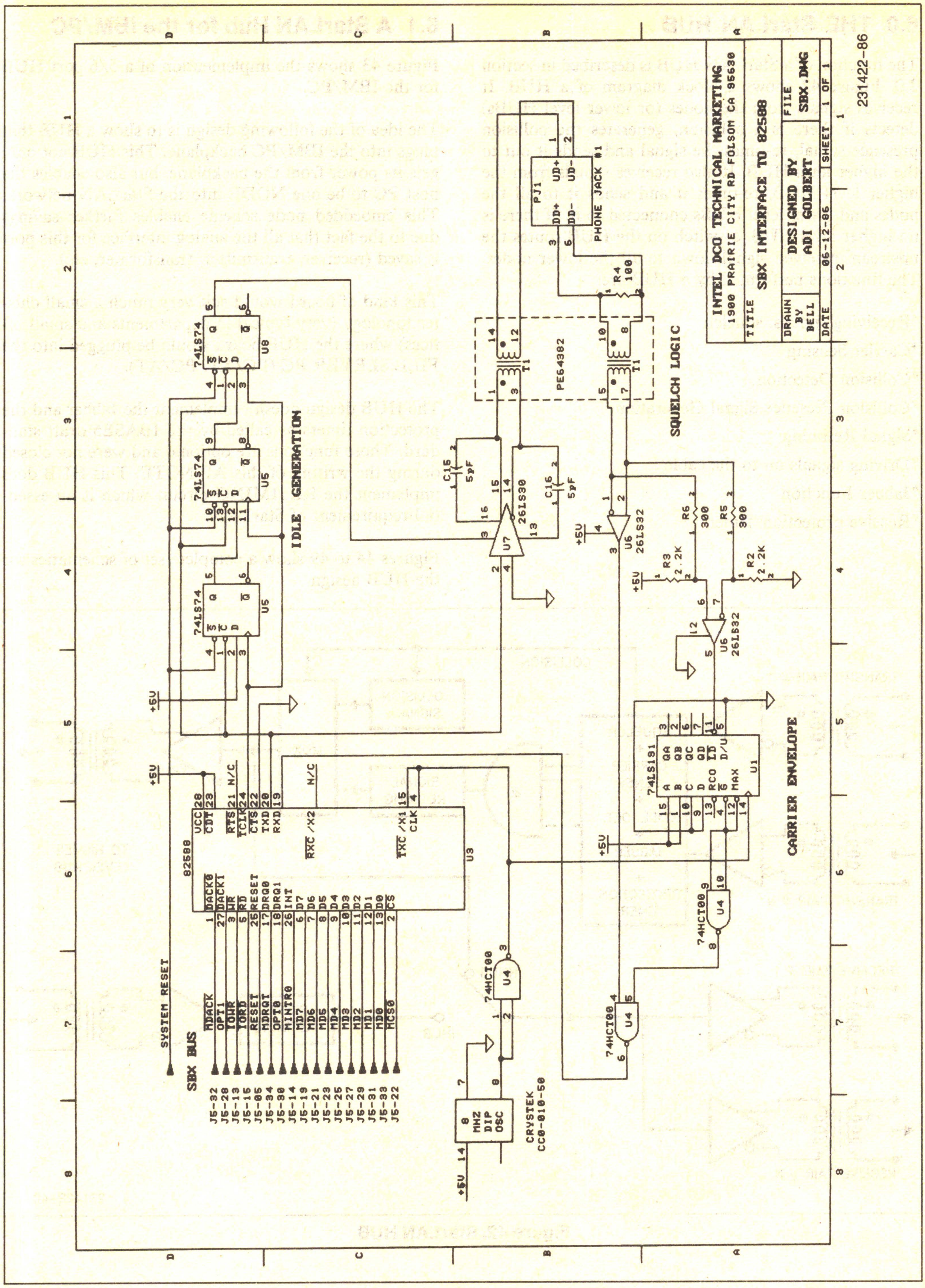


Figure 41. iSBX Interface to 82588

6.0 THE StarLAN HUB

The function of a StarLAN HUB is described in section 2.0. Figure 42 shows a block diagram of a HUB. It receives signals from the nodes (or lower level HUBs) detects if there is a collision, generates the collision presence signal, re-times the signal and sends it out to the higher level HUB. It also receives signals from the higher level HUB, re-times it and sends it to all the nodes and lower level HUBs connected to it. If there is no higher level HUB, a switch on the HUB routes the upstream received signal down to all the lower nodes. The functions performed by a HUB are:

*Receiving signals, squelch

*Carrier Sensing

*Collision Detection

*Collision Presence Signal Generation

*Signal Retiming

*Driving signals on to the cable

*Jabber Function

*Receive protection Timer

6.1 A StarLAN Hub for the IBM/PC

Figure 43 shows the implemention of a 5/6 port HUB for the IBM/PC.

The idea of the following design is to show a HUB that plugs into the IBM/PC backplane. This HUB not only gets its power from the backplane, but also enables the host PC to be one NODE into the StarLAN network. This embedded node scheme enables further savings due to the fact that all the analog interface for this port is saved (receiver, transmitter, transformer, etc).

This kind of board would suit very much a small cluster topology (very typical in departments and small offices) where the HUB board would be plugged into the FILE SERVER PC (PC/XT, PC/AT).

The HUB design doesn't implement the Jabber and the protection timers as called by the 1BASE5 draft standard. Those functions are optional and were not closed during the writing of this AP-NOTE. This HUB does implement the RETIMING circuit which is an essential requirement of StarLAN.

Figures 44 to 49 show a complete set of schematics for the HUB design.

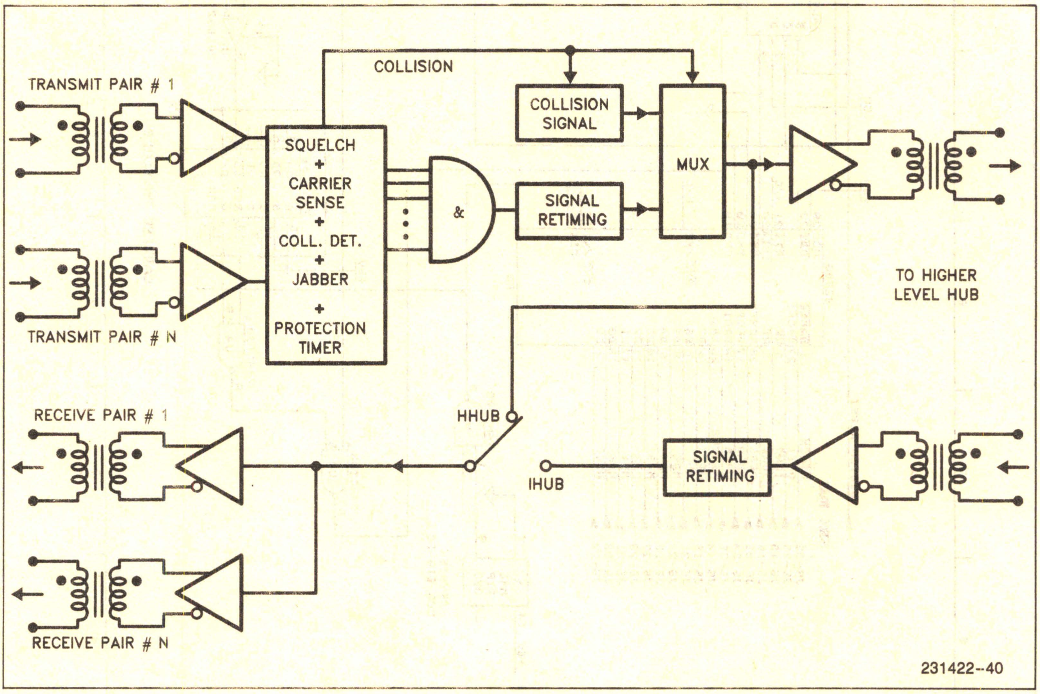
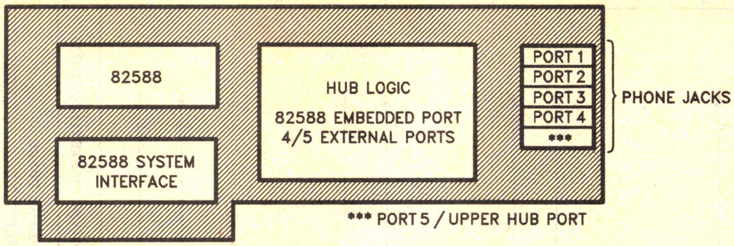


Figure 42. StarLAN HUB



231422-87

- Low Cost HUB, Uses IBM/PC Power Supply
- 82588, Embedded Port Savings
  - Transformers
  - 422 Drivers
- Functional StarLAN Cluster, For Low Cost/Small Topologies

Figure 43. IBM/PC Resident HUB

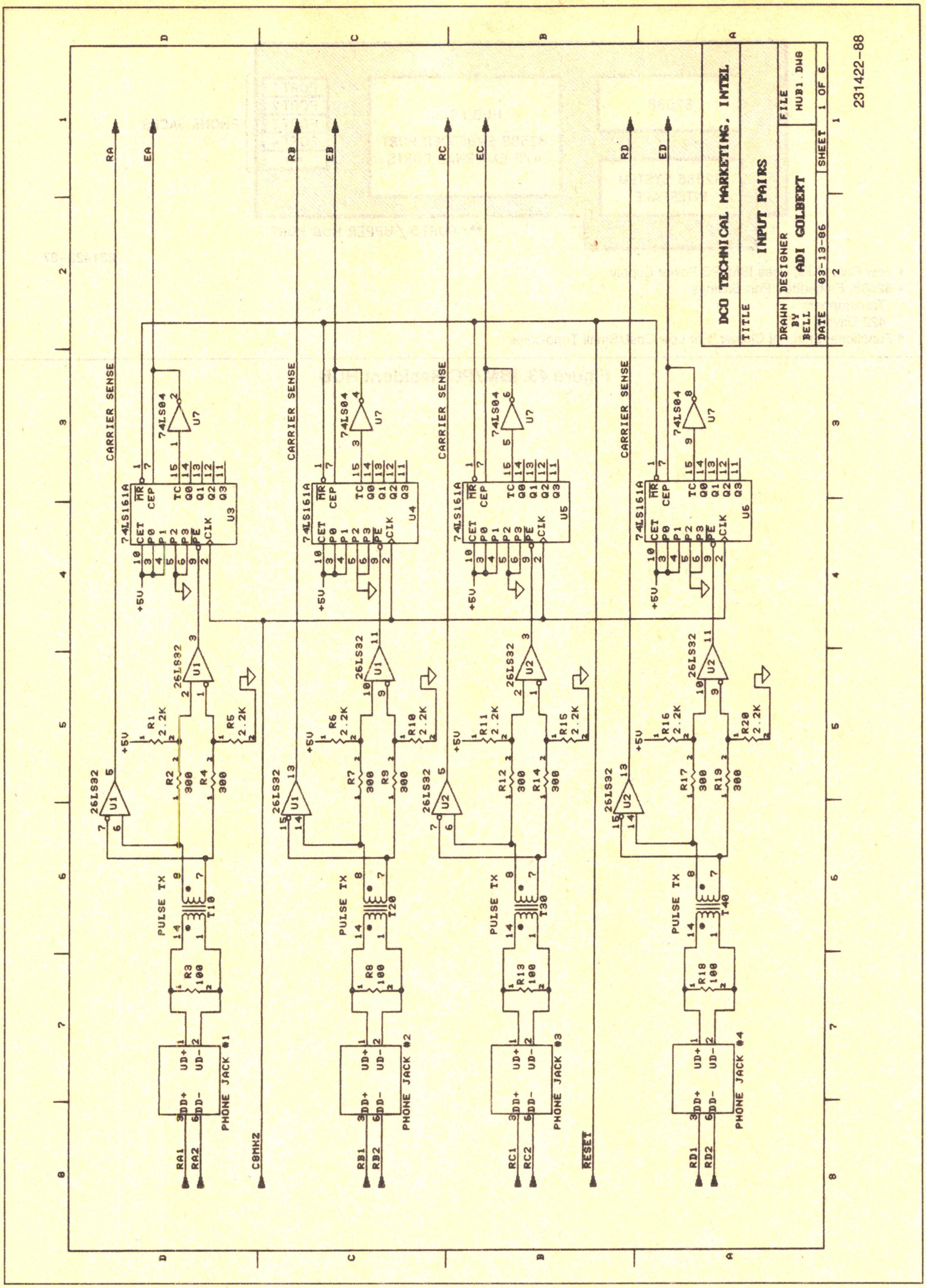

DCO TECHNICAL MARKETING, INTEL
TITLE
INPUT PAIRS
DRAWN BY BELL
DESIGNER ADI GOLBERT
DATE 03-13-86
FILE HUB1.DWG
SHEET 1 OF 6
231422-88
CARRIER SENSE
74LS161A
74LS04
26LS32
PULSE TX
PHONE JACK #1
PHONE JACK #2
PHONE JACK #3
PHONE JACK #4
RA
EA
RB
EB
RC
EC
RD
ED
RA1
RA2
RB1
RB2
RC1
RC2
RD1
RD2
C8MHZ
RESET

Figure 44

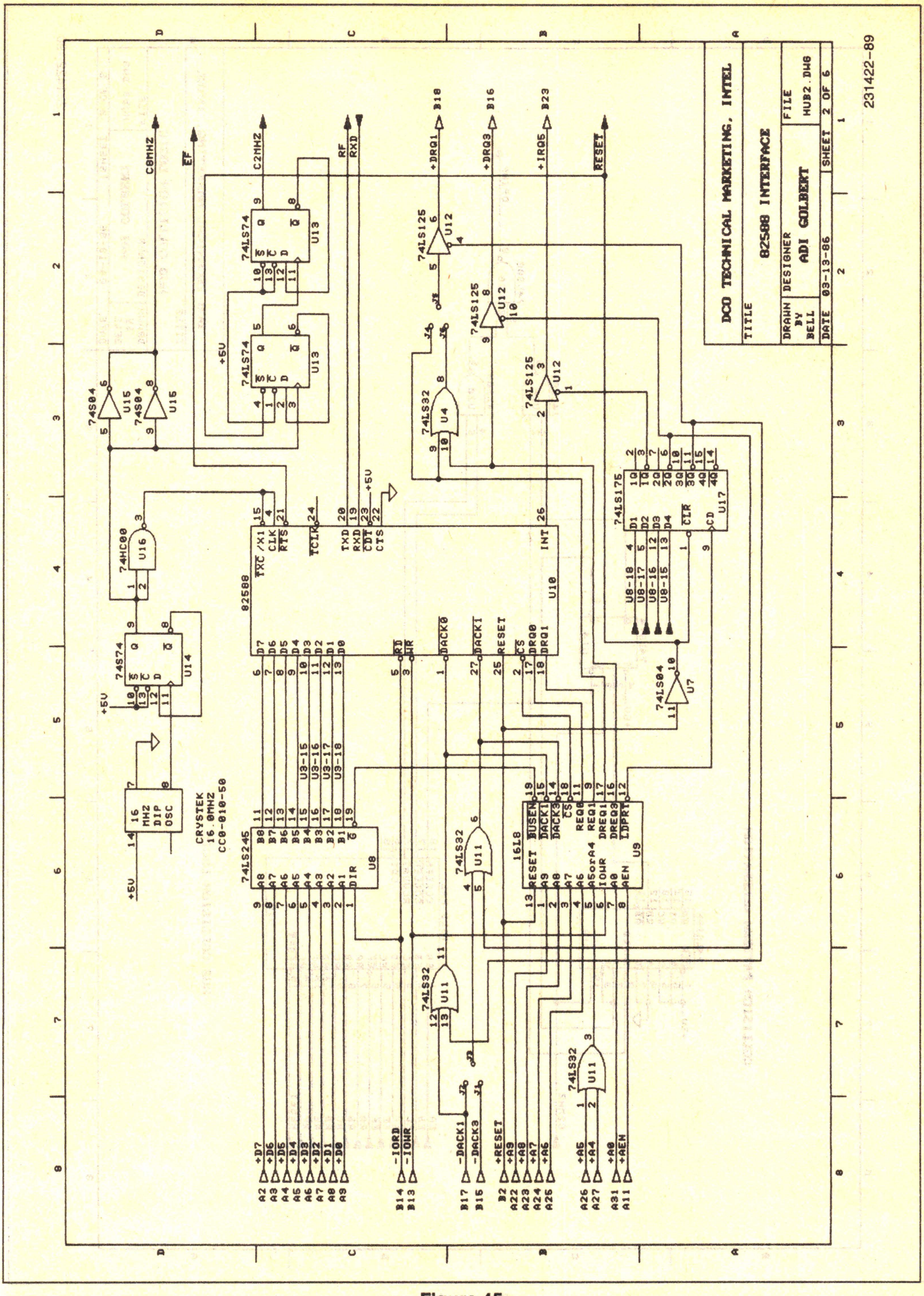

DCO TECHNICAL MARKETING, INTEL
82588 INTERFACE
DESIGNER ADI GOLBERT
DRAWN BY BELL
DATE 03-13-86
FILE HUB2.DWG
SHEET 2 OF 6
231422-89

Figure 45

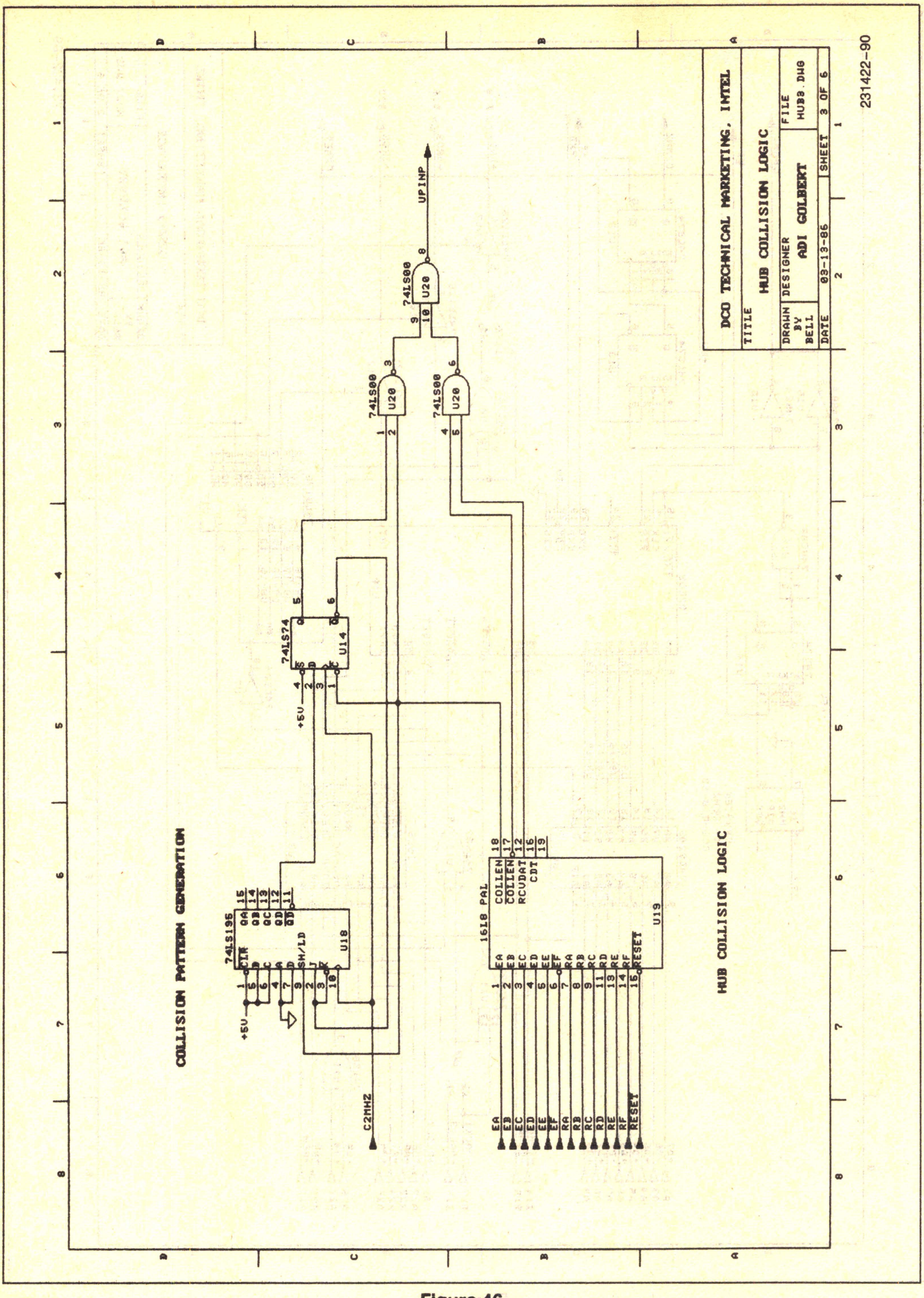

DCO TECHNICAL MARKETING, INTEL
TITLE
HUB COLLISION LOGIC
DRAWN BY BELL
DESIGNER
ADI GOLBERT
FILE
HUB3.DWG
DATE
03-13-86
SHEET
3 OF 6
COLLISION PATTERN GENERATION
HUB COLLISION LOGIC
74LS195
U18
74LS74
U14
74LS00
U20
1618 PAL
U19
COLLEN
RCVDAT
CDT
UPINP
C2MHZ
+5V
EA
EB
EC
ED
EE
EF
RA
RB
RC
RD
RE
RF
RESET
231422-90

Figure 46

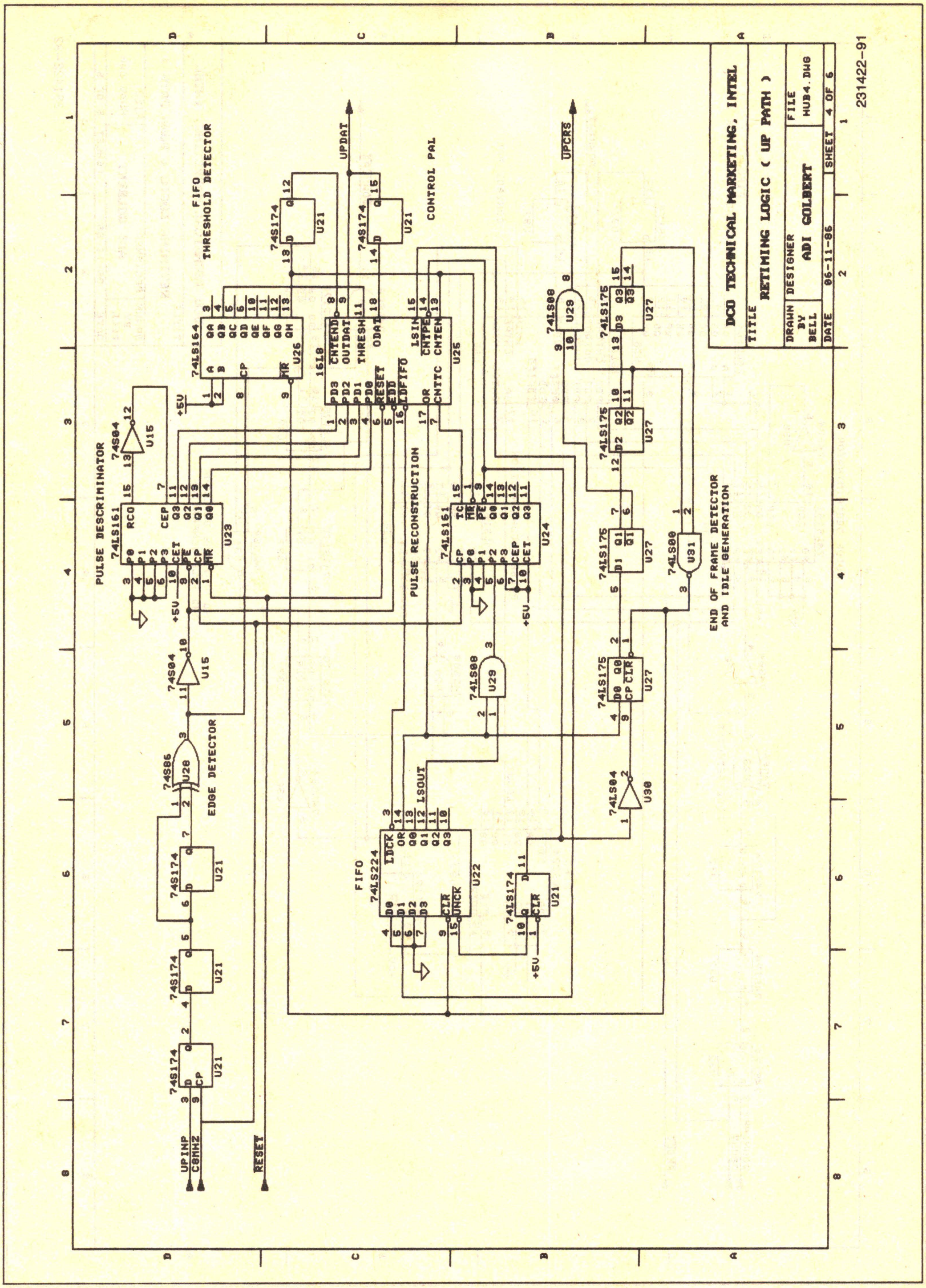

DCO TECHNICAL MARKETING, INTEL
TITLE
RETIMING LOGIC (UP PATH)
DRAWN BY BELL
DESIGNER ADI COLBERT
FILE HUB4.DWG
DATE 06-11-86
SHEET 4 OF 6
231422-91
FIFO THRESHOLD DETECTOR
CONTROL PAL
PULSE DESCRIMINATOR
PULSE RECONSTRUCTION
EDGE DETECTOR
FIFO
END OF FRAME DETECTOR AND IDLE GENERATION
UPDAT
UPCRS
UPINP
C8MHZ
RESET
LSOUT
74LS164
74LS161
74S174
74S04
74S86
74LS224
74LS175
74LS08
16L8
U21
U22
U23
U24
U25
U26
U27
U28
U29
U30
U31
+5V

Figure 47

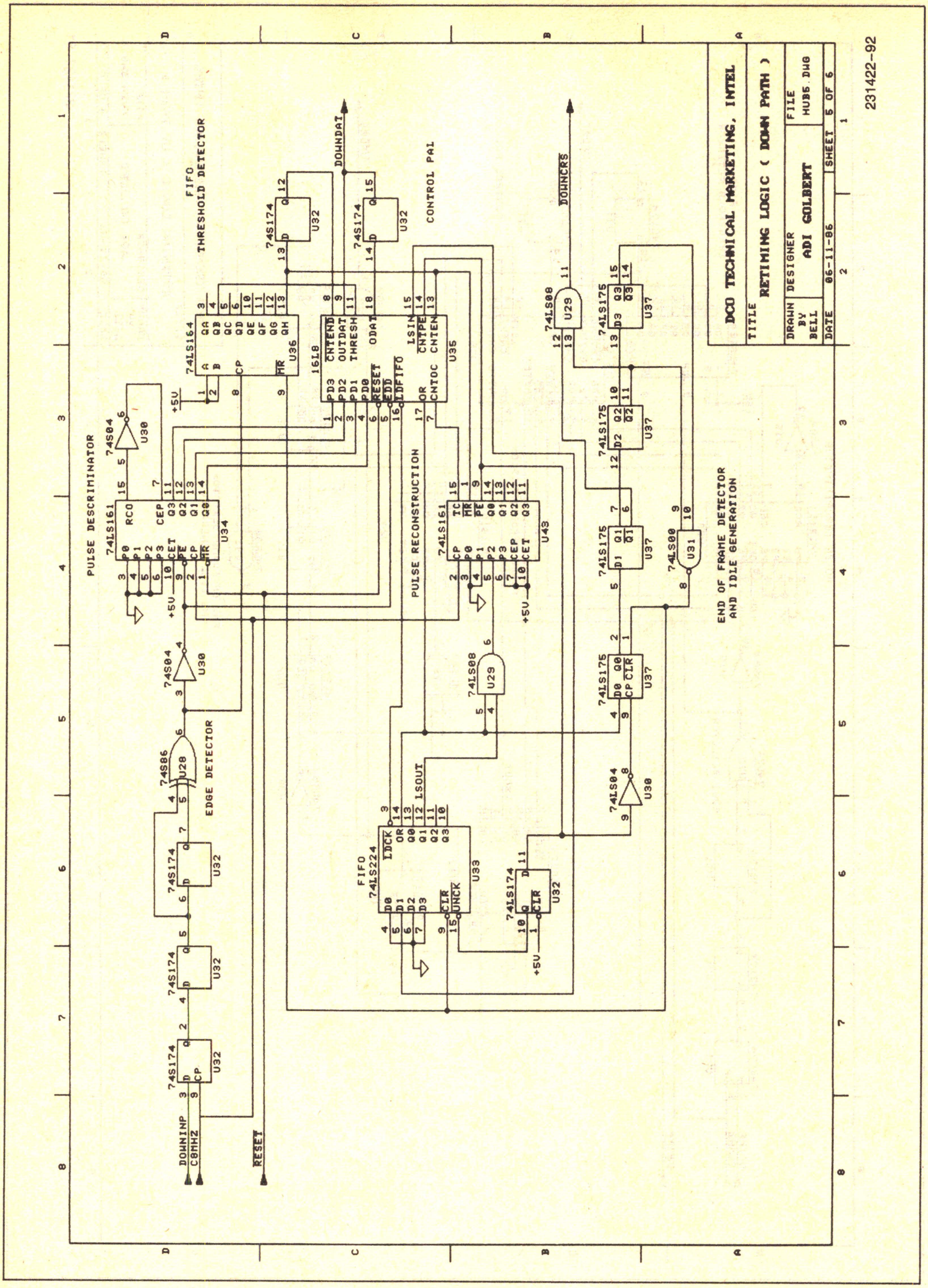
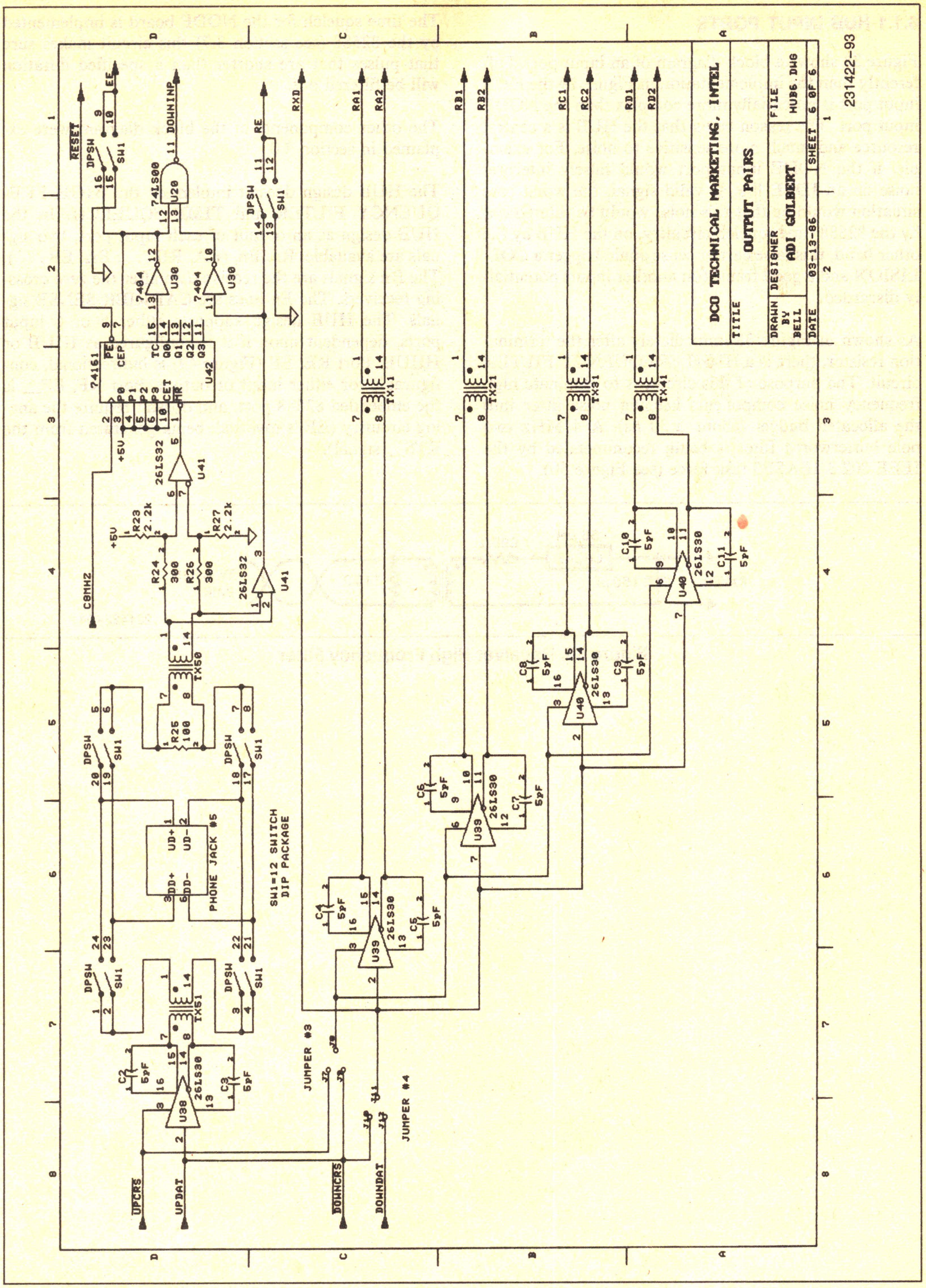

DCO TECHNICAL MARKETING, INTEL
TITLE
RETIMING LOGIC (DOWN PATH)
DRAWN BY BELL
DESIGNER ADI GOLBERT
FILE HUBS.DWG
DATE 06-11-86
SHEET 5 OF 6
231422-92
FIFO THRESHOLD DETECTOR
CONTROL PAL
PULSE DESCRIMINATOR
PULSE RECONSTRUCTION
EDGE DETECTOR
END OF FRAME DETECTOR AND IDLE GENERATION
FIFO
74LS224
74LS164
16L8
74LS161
74S174
74LS175
74S86
74S04
74LS04
74LS08
74S08
DOWNDAT
DOWNCRS
DOWNINP
C8MHZ
RESET
LSOUT
+5V

Figure 48



231422-93

Figure 49

6.1.1 HUB INPUT PORTS

Figure 38 shows a block diagram of an input port. Differently than the implementation in Figure 29 the HUB input port is potentially more complex than the NODE input port. The reason being that the HUB is a central resource and much more sensitive to noise. For example, if the NODE input port would falsely interpret noise on an IDLE line as valid signal, the worst case situation would be that this noise would be filtered out by the 82588 time squelch circuitry, on the HUB by the other hand, this false carrier sense could trigger a COLLISION and a good frame (on another input) potentially discarded.

As shown in Figure 38 immediately after the termination resistor, there is a HIGH FREQUENCY FILTER circuit. The purpose of this circuit is to eliminate high frequency noise components keeping noise jitter into the allocated budget (about ±30 ns). A 4 MHz two pole butterworth filter is being recommended by the IEEE 802.3 1BASE5 task force (see Figure 50).

The time squelch for the NODE board is implemented by the 82588 (see section 3.7) this circuit makes sure that pulses that are shorter than a specified duration will be filtered out.

The other components of the block diagram were explained in section 3.0.

The HUB design doesn't implement the HIGH FREQUENCY FILTER and TIME SQUELCH. In the HUB design as an output of each input port, two signals are available: Rn, En, (RA, RB . . . , EA, EB . . .). The Rn signals are the receive data after the zero crossing receivers. The En lines are CARRIER SENSE signals. The HUB design supports either 5 or 6 input ports, dependent upon if it is configured as IHUB or HHUB. Port RE, EE (Figure 49) is bidirectional, configurable for either input or output. Port RF, EF__ is the embedded 82588 port, and doesn't require the analog circuitry (EF is inverted, being generated from the RTS__ signal).

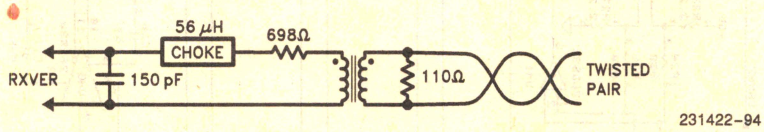


Figure 50. Receiver High Frequency Filter

6.1.2 COLLISION DETECTION

Rn and En signals from each channel are fed to a 16L8 PAL, where the collision detection function is performed.

Collision Detection in the StarLAN HUB is performed by detecting the presence of activity on more than one input channels. This means if the signal En is active for more than one channel, a collision is said to occur. This translates to the PAL equations:

COLLISION DETECTION:

```
CDT = ! (EA & !EB & !EC & !ED & !EE & EF__ #          (only EA active)
        ! EA & EB & !EC & !ED & !EE & EF__ #          (only EB active)
        ! EA & !EB & EC & !ED & !EE & EF__ #          (only EC active)
        ! EA & !EB & !EC & ED & !EE & EF__ #          (only ED active)
        ! EA & !EB & !EC & !ED & EE & EF__ #          (only EE active)
        ! EA & !EB & !EC & !ED & !EE & !EF__ #         (only EF active)
        ! EA & !EB & !EC & !ED & !EE & EF__ );   (none of the inputs active)
```

COLLISION DETECTION SR-FF:

```
COLLEN__ = ! (CDT # COLLEN );                        (set with collision)

COLLEN__ = ! ( RESET__ # COLLEN__ #
             ( !CDT & !EA & !EB & !EC & !ED & !EE & EF__);

                                         ( reset when all inputs inactive )
```

RECEIVE DATA OUTPUT:

```
RCVDAT = (   ( RA # !EA ) & ( RB # !EB ) & ( RC # !EC ) &

             ( RD # !ED ) & (RE # !EE ) & (RF # EF__ ) );

                                          ( output is high if no active input)
```

The COLLEN signal once triggered will stay active until all inputs go quiet. This signal is used externally to either enable passing RCVDAT or the collision presence signal (CPS) to the retiming logic. An external multiplexer using 3 nand gates is used for this function. Note that in this specific implementation the CPS/RCVDAT multiplexer is before the retiming logic, which is different from Figure 42 diagram. StarLAN provides enough BIT-BUDGET delay to allow the CPS signal to be generated through the retiming FIFO. In this HUB implementation it was decided to use this option to make sure that the CPS startup is synchronized with the previously transmitted bit as required by the 1BASE5 draft.

6.1.3 THE LOCAL 82588

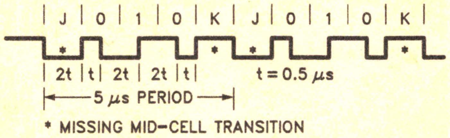
As described before, the purpose of the local 82588 is to enable the Host IBM/PC to also be a node into the StarLAN network. The interface of this 82588 is exactly similar to the one explained in section 5. The RTS__ signal serves as the carrier EF__ signal, and TXD as RF signal. This local node interfaces to the HUB without any analog interface which is a significant saving.

6.1.4 THE COLLISION PRESENCE SIGNAL

The Collision Presence Signal (CPS) is generated by the HUB whenever the HUB detects a collision. It then propagates the CPS to the higher level HUB. The CPS signal pattern is shown in Figure 51. Whenever a StarLAN node receives this signal, it should be able to detect within a very few bit times that a collision occurred. Since the nodes detect the occurrence of a collision by detecting violations in Manchester encoding, the CPS must obviously be a signal which violates Manchester encoding. Section 3.5 shows that the CPS has missing mid-cell transitions occurring every two and a half bit cells. These are detected as Manchester code violations. Thus, the StarLAN node is presented with collision detection indications every two and a half ms. This results in fast and reliable detection of collisions. CPS has a period of 5 ms.

One may wonder why such a strange looking signal was selected for CPS. The rationale is that this CPS looks very much like a valid Manchester signal—edges are 0.5 or 1.0 microsec. apart—resulting in identical radiation, cross-talk and jitter characteristics as a true Manchester. This also makes the re-timing logic for the signals simpler—it need not distinguish between valid Manchester and CPS. Moreover, this signal is easy to generate.

A few important requirements for CPS signal are: a) it should be generated starting synchronized with the last transmitted bit cell. CPS is allowed to start either low or high, but no bit cell of more than 1 microsecond is allowed (Avoid false idles, very long "low" bits). b) once it starts, it should continue until all the input lines to the HUB die out. Typically, when the collision occurs, the multiplexor in the HUB switches from RCV signal to the CPS. This switch is completely asynchronous to the currently being transmitted data, and by such may violate the requirement of not having bit cells longer than 1 μs. In order to avoid those long pulses, the output of the CPS/RCVDAT multiplexer is passed through the retiming circuitry which will correct those long pulses to their nominal value. The reason for restriction b) is to ensure that the CPS is seen by all nodes on the network since it is generated until every node has finished generating the Jam pattern.



231422-42

- Collision Presence Signal (CPS) is generated by the HUB when it detects more than one input line active.
- CPS violates Manchester encoding rules—due to missing mid-cell transitions—hence is detected as a collision by the DTE (82588).

Choice of Collision Presence Signal

- It is a Manchester look-alike signal—edges are 0.5 or 1.0 μs apart.
  - — Identical radiation, crosstalk and jitter characteristics
  - — Eases retiming of the signal in the HUB
- It is easy to generate—1.5 TTL pack, or in a PAL

Figure 51. Collision Presence Signal

CPS is generated using a 4-bit shift register and a flip-flop as shown in Figure 52. It works off a 2 MHz clock. A closer look at the CPS waveform shows that it is inverse symmetric within the 5 μs period. The circuit is a 5-bit shift register with a complementary feedback from the last to the first bit. The bits remain in defined states (01100) till collision occurs. On collision the bits start rotating around generating the pattern of 0011011001, 0011011001, 00110 ... with each state lasting for 0.5 μs.

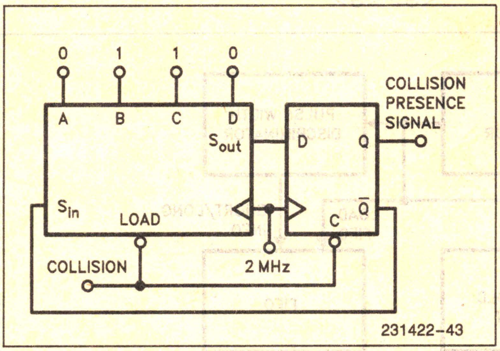


Figure 52. Collision Presence Signal Generation

6.1.5 SIGNAL RETIMING

Whenever the signal goes over a cable it suffers jitter. This means that the edges are no longer separated by the same 0.5 or 1.0 μs as at the point of origin. There are various causes of jitter. Drivers, receivers introduce some shifting of edges because of differing rise and fall times and thresholds. A random sequence of bits also produces a jitter which is called intersymbol interference, which is a consequence of different propagation delays for different frequency harmonics in the cable. Meaning short pulses have a longer delay than long ones. A maximum of 62.5 ns of jitter can accumulate in a StarLAN network from a node to a HUB or from a HUB to another HUB. The following values show what are the jitter components:

Transmitter skew	±10 ns
Cable Intersymbol interference	±9 ns
Cable Reflections	±8 ns
Reflections due to receiver termination mismatch	±5 ns
HUB fan-in, fan-out	±5 ns
Noise	±25.5
Total	±62.5 ns

It is important for the signal to be cleaned up of this jitter before it is sent on the next stretch of cable because if too much jitter accumulates, the signal is no longer meaningful. A valid Manchester signal would, as a result of jitter, may no longer be decodable. The process of either re-aligning the edges or reconstructing the signal or even re-generating the signal so that it once again "looks new" is called re-timing. StarLAN requires for the signal to be re-timed after it has travelled on a segment of cable. In a typical HUB two re-timing circuits are necessary; one for the signals going upstream towards the higher level HUB and the other for signals going downstream towards the nodes.

6.1.6 RETIMING CIRCUIT, THEORY OF OPERATION

This section will discuss the principles of designing a re-timing circuit. Figure 53 shows the block diagram of a re-timing circuit. The data coming in is synchronized using an 8 MHz sampling clock. Edges in the waveform are detected doing an XOR of two consecutive samples. A counter counts the number of 8 MHz clocks between two edges. This gives an indication of long (6 to 10 clocks) or short (3 to 5 clocks) pulses in the received waveform. Pulses shorter than 3 clocks are filtered out. Every time an edge occurs, the length—(S)hort or (L)ong—of the pulse is fed into the FIFO. Retiming of the waveform is done by actually generating a new waveform based on the information being pumped into the FIFO. The signal regeneration unit reads the FIFO and generates the output waveform out of 8 MHz clock pulses based on what it reads, either short or longs. In summary every time a bit is read from the fifo, it indicates that a transition needs to occur, and when to fetch the next bit. When idle the output of the retiming logic starts with a "high" level.

FIFO	Output
empty	1111
S	0000
S	1111
L	00000000
L	11111111

It can be seen that the output always has edges separated by 4 or 8 clock pulses—0.5 or 1.0 μs.

The FIFO is primarily needed to account for a difference of clock frequencies at the source and regeneration end. Due to this difference, data can come in faster or slower than the regeneration circuit expects. A 16 deep FIFO can handle frequency deviations of up to 200 ppm for frame lengths up to 1600 bytes. The FIFO also overcomes short term variations in edge separation. It is essential that the FIFO fills in up to about half before the process of regeneration is started. Thus, if the regeneration is done at a clock slightly faster than the source clock, there is always data in the FIFO to work from. That is why the FIFO threshold detect logic is necessary, which counts 8 edges and then enables the signal regeneration logic.

Example:

Input Waveform ...11110001111000000011111111110001111100...

Input into the FIFO <S> <S> <L> <L><S> <S>

Regenerated Output:

Output: ...111110000111100000000111111110000111...

FIFO: <S> <S> <L> <L> <S> <S>

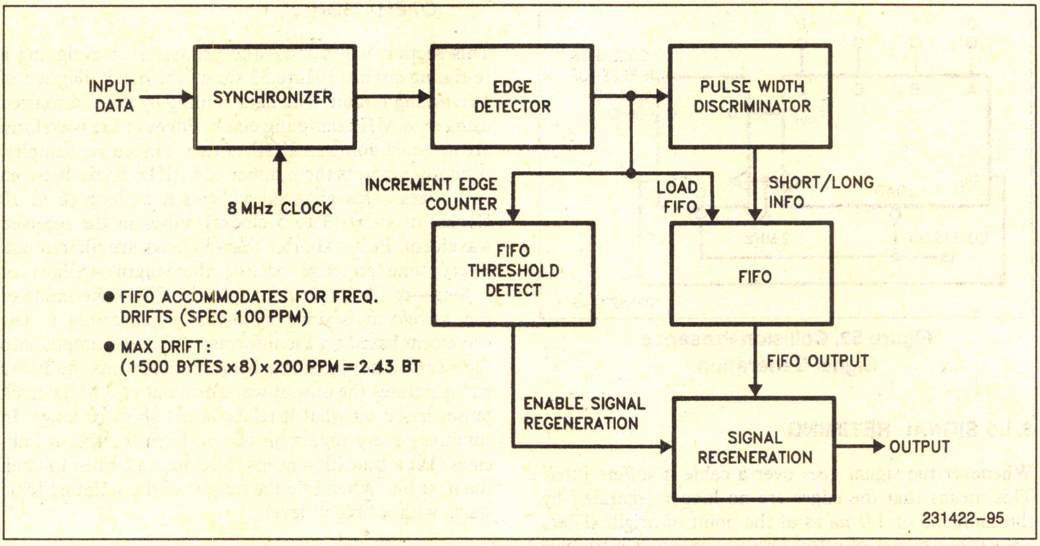


Figure 53. Retiming Block Diagram

6.1.7 RETIMING CIRCUIT IMPLEMENTATION

The retiming circuit implementation can be seen in Figures 47, 48. Both figures implement exactly the same function, one for the upstream, and the other for the downstream. The retiming circuit was implemented using about 8 SSI, MSI TTL components, one fifo chip and one PAL. The purpose of implementing this function with discrete components was to show the implementation details. The discussion of the implementation will refer to Figure 47 for unit numbers.

The signal UPIMP which is an output of the HUB multiplexing logic, is asynchronous to the local clock. This signal is synchronized by two flip-flops and fed into an edge generation logic (basically an XOR gate that compares the present sample with the previous one). On every input transition a 125 ns pulse will be generated at the output of the edge detector (U28). This pulse will reset the 74LS161 counter that is responsible for measuring pulse widths (in X8 clock increments). The output of the pulse discriminator will reflect the previous pulse width every time a new edge is detected. The following events will take place on every detected edge:

1. U26 which is the threshold detector will shift one "1" in. The outputs of U26 will be used by the control PAL to start the reconstruction process.
2. The output of U23 which specifies the last pulse width will be input into the control PAL for determining if it was a long or short pulse. The result of this evaluation will be the LSIN signal which will be loaded into the fifo (U22).

U22 is the retiming FIFO, it is 16x4 fifo, but only one bit is necessary to store the SHORT/LONG information.

CONTROL LOGIC PAL functions (U25):

Signals definition:

INPUTS:

PD0..PD3: Outputs of the pulse descriminator, indicate the width of the last measured pulse.

EDD_: Output of the edge detector, pulse of 125 ns width, indicates the occurrence of an edge in the input data.

THRESH: Output of the threshold logic, indicates at least one bit was already received.

CNTEN: Output of the Threshold logic, indicates 7 bits have been loaded into the FIFO, and that signal reconstruction can begin.

CNTEND: The same signal as before delayed by one clock.

OUTDAT: Output of the retiming logic, is feedback into the PAL to implement a clocked T-FF.

RESET_: Resets the retiming logic.

CNTTC: Terminal count of the reconstruction counter, indicating that reconstruction of a new bit will get started.

OR: Output of the FIFO indicating, that the FIFO is empty and that IDLE generation can get started.

OUTPUTS:

LDFIFO_: Loads SHORT/LONG indications into the FIFO.

LSIN: Indicates SHORT/LONG

CNTPE_: Loads FIFO SHORT/LONG output into the reconstruction counter.

ODAT: Together with the external U21 flip-flop and OUTDAT implement a clocked T-FF.

Loading the FIFO will be done every time there is an edge, we have passed the one bit filter threshold level, and the pulse width is longer than two 8X clocks. This one bit threshold level serves as a time domain filter discarding the first received preamble bit.

```
LDFIFO_ = ! ( PD1 # PD2 # PD3 ) & !EDD_ & THRESH ) ;

Whenever there is an edge, we are above the first received bit threshold
and the pulse width is longer than "1" the fifo is loaded.

LSIN    = ! (PD3 # (PD2 & PD0) # (PD2 & PD1)) ;

Every pulse longer than 6 is considered to be a long pulse.

CNTPE_  = ! ( (CNTEN & !CNTEND) # CNTTC ) ;

The reconstruction counter is loaded in two conditions:

Whenever CNTEN comes active, meaning the FIFO threshold of seven was exceeded.
Whenever the terminal count of U24 is active meaning a new pulse is going to be reconstructed.

ODAT    = !RESET_ # ( !CNTPE_ & !OUTDAT)   (A)
                  # (  CNTPE_ &  OUTDAT)   (B)
                  # ( !CNTPE_ & !OR)       (C)

Minterm (A) and (B) implement a T-FF, whenever CNTPE_ is "low"
ODAT will toggle. The external U21 is part of this flip-flop.
Minterm (C) insures the output of the flip-flop will go inactive
"high" when the FIFO is empty. RESET_ causes the output to go
"high" on initialization.
```

U24 as mentioned is the reconstruction counter. This counter is loaded by the control logic with either 8 or 12, it counts up and is reloaded on terminal count. Essentially generating at the output nominal length longs and shorts.

U22 is the retiming FIFO, and its function as mentioned is to accommodate frequency skews between the incoming and outgoing signal.

U27 is the IDLE generation logic. The purpose of this logic is to detect when the FIFO is empty, meaning that no more data needs to be transmitted. On detection of this event this component will generate 2 ms of IDLE time. On the end of IDLE the whole retiming logic will be reset.

6.1.8 DRIVER CIRCUITS

The signal coming out of the RETIMING LOGIC is fed into 26LS30s and pulse transformers to drive the twisted pair lines (See section 5.0 for details).

6.1.9 HEADER/INTERMEDIATE HUB SWITCH

As seen on Figure 43 this hub can be configured as either an intermediate hub, or a Header one. One of the phone jacks, more specifically JACK #5 is either an input port or an output one. In order to implement this function, an 8 position DIP SWITCH (SW1) is used. The phone jacks are marked with UD, DD notation, meaning upstream data, and downstream data respectively. As specified in the StarLAN 1BASE5 draft NODES transmit data on UD pair, and HUBS on the DD pair. Switch SW1 has the function to invert UD, DD in PHONE JACK #5 to enable it to be either input or output port.

6.1.10 JABBER FUNCTION

This design does not implement the jabber unit but it is described here for completeness. IEEE 802.3 does not mandate this feature, but it is "Strongly Recommended". The jabber function in the HUB protects the network from abnormally long transmissions by any node.

Two timers T1, T2 are used by the JABBER function. They may be implemented either as local timers (one for each HUB port) or as global timers shared by all ports. After detecting an input active, timers T1, T2 will be started, and T1 will time out after 25 to 50 ms. T2 will time-out after 51 to 100 ms. During T2 time, after T1 expired, the HUB will send the CP-PATTERN informing any jamming stations to quit their transmissions. If on T2 time-out there are still jamming ports, their input is going to be disabled. A disabled port, will be reenabled whenever its input becomes again active and the downward side is idle.

The following is an explanation of the requirement that the downward side be idle to reenable an input port. Consider the case of Figure 54. The figure shows a two port HUB. Port A has two wires A_u, A_d for the up and down paths. Port B has B_u, B_d respectively. Port C is the output port, that broadcasts to the other HUBs higher in the hierarchy. Consider the case as shown, where B_u and B_d are shorted together. Suppose the case that port A_u is active. Its signal will propagate up in the hierarchy through C_u and come down from C_d to A_d, and B_d. Due to the short between B_d and B_u the signal will start a loop, that will first cause a collision and jam the network forever. This kind of fault is taken care of by the jabber circuitry. T1 and T2 will expire, causing the jabber logic to disable B_u input. Upon this disabling B_u is going to go Idle and be a candidate for future enabling. Suppose now that A_u is once again active. If the reenable condition would not require C_d to be IDLE, B_u would be reenabled causing the same loop to happen once again. Note that in this case C_d will be active before B_u causing this port to continue to be disabled and avoiding the jamming situation (Figure 55) gives a formal specification of the jabber function).

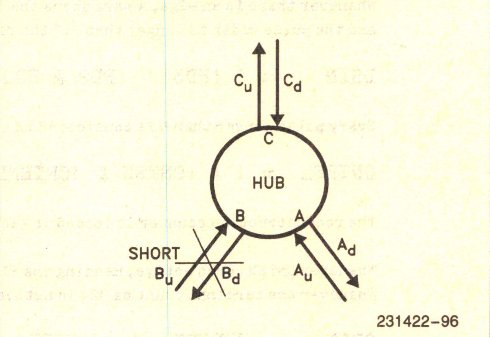


Figure 54. Jabber Function

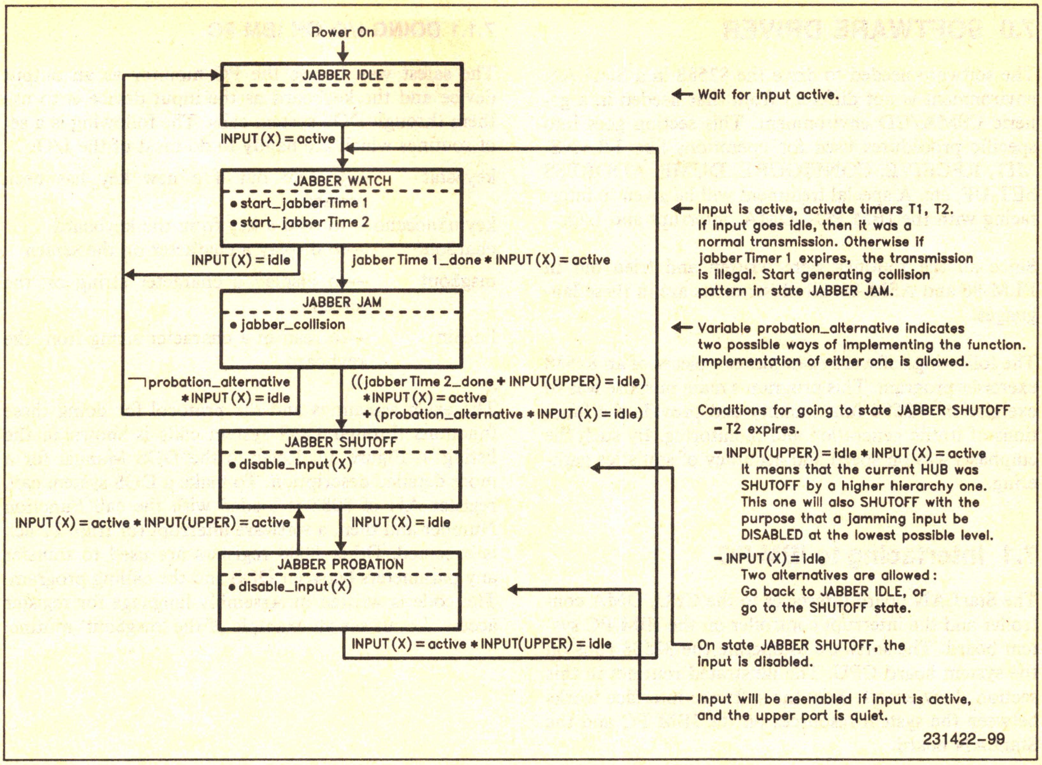


Figure 55. Jabber State Diagram

6.1.11 HUB RECEIVER PROTECTION TIMER

On the end of a transmission, during the transition from IDLE to high impedance state, the transmitter will exhibit an undershoot and/or ringing, as a consequence of transformer discharge. This undershoot/ringing will be transmitted to the receiver which needs to protect itself from false carriers due to this effect. One way of implementing this protection mechanism is to implement a blind timer, which upon IDLE detection will "blind" the receiver for a few microseconds.

Causes of the transmitter undershoot/ringing:

1. Difference in the magnitudes of the differential output voltage between the high and the low output stages.
2. Waveform assymmetry due to transmitter jitter.
3. Transmitter and receiver inductance (transformer L).
4. Two to three microseconds of IDLE pattern.

All the described elements will contribute to energy storage into the transformer inductor, which will discharge during the transition of the driver to high impedance.

The blinding timer is currently defined to be from 20 to 30 microseconds for the HUBs, being from 0 to 30 microseconds for the nodes (optional). The 82588 has built-in this function. It won't receive any frames for an inter-frame-spacing (IFS) from the idle detection.

6.1.12 HUB RELIABILITY

Since the StarLAN HUBs form focal points in the network, it is important for them to be very reliable, since they are single points of failure which can affect a number of nodes or can even bring down the whole network. StarLAN 1BASE5 draft requires HUBs to have a mean time between failures (MTBF) of at least 5 years of continuous operation.

7.0 SOFTWARE DRIVER

The software needed to drive the 82588 in a StarLAN environment is not different from that needed in a generic CSMA/CD environment. This section goes into specific procedures used for operations like TRANSMIT, RECEIVE, CONFIGURE, DUMP, ADDRESS SET-UP, etc. A special treatment will be given to interfacing with the IBM PC—DMA, interrupt and I/O.

Since all the routines were written and tried out in PLM-86 and ASM-86, all illustrations are in these languages.

The following software examples are pieces of an 82588 exerciser program. This program's main purpose was to exercise the 82588 functionality and provide the functions of traffic generation and monitoring. By such the emphasis was on speed and accuracy of statistics gathering.

7.1 Interfacing to IBM PC

The StarLAN board interfaces to the CPU, DMA controller and the interrupt controller on the IBM PC system board. The software to operate the 82588 runs on the system board CPU. The illustrated routines in this section show exactly how the software interface works between the system resources on the IBM PC and the StarLAN board.

7.1.1 DOING I/O ON IBM PC

The safest way to use the PC monitor as an output device and the keyboard as the input device is to use them through DOS system calls. The following is a set of routines which are handy to do most of the I/O:

key$stat	—to find out if a new key has been pressed
keyin$noecho	—to read a key from the keyboard
char$out	—to display a character on the screen
msg$out	—to display a character string on the screen
line$in	—to read in a character string from the keyboard

The exact semantics and the protocol for doing these functions through DOS system calls is shown in the listing in Figure 56. Refer to the DOS Manual for a more detailed description. To make a DOS system call, register AH of 8088 is loaded with the call Function Number and then, a software interrupt (or trap) 21 hex is executed. Other 8088 registers are used to transfer any parameters between DOS and the calling program. The code is written in Assembly language for register access. Let us see an example of the 'msg$out' routine:

```
lds dx,STRING_POINTER      ; load pointer to string in reg. ds:dx
mov ah,09h                 ; 9 = function number for string o/p
int 21h                    ; DOS System Call
```

These procedures are called from another module, written in a higher level language like PLM-86. The parameters are transferred to the ASM-86 routines on the stack.

Examples of using the I/O routines:

```
KEY_STATUS = key$stat;                        /* INQUIRE KEYBOARD STATUS    */
NEW_KEY = keyin$noecho;                       /* INPUT NEW KEY              */
call line$in(@LINE_BUFFER);                   /* STRING INPUT               */
call char$out(CHAR_OUT);                      /* TO OUTPUT CHAR_OUT ON SCREEN*/
call msg$out(@('THIS IS A MESSAGE.$'));       /* OUTPUT STRING              */
                                              /* NOTE $ TERMINATOR          */
```

```
/*-------------------------------------------------------------------------*/
/*     Declarations for external IBM PC I/O routines                       */
/*-------------------------------------------------------------------------*/

key$stat: procedure byte external;          /* key status routine */
end key$stat;

key$in$noecho: procedure byte external;        /* console input routine */
end key$in$noecho;

char$out: procedure(char) external;    /* console output routine */
declare char byte;
end char$out;

msg$out: procedure(msg$ptr) external;          /* console string output routine
declare msg$ptr pointer;
end msg$out;

line$in: procedure(line$ptr) external;         /* console string input routine
declare line$ptr pointer;
end line$in;


Assembly Language implementation of the routines

$TITLE(IBM/PC  DOS CALLS PROCEDURES)

            NAME        DOSPROCS
;
DGROUP                  GROUP   DATA
CGROUP                  GROUP   CODE
;
DATA                    SEGMENT WORD PUBLIC 'DATA'
DATA                    ENDS
;
DOS         EQU         21H
;
CODE                    SEGMENT WORD PUBLIC 'CODE'
                        ASSUME  CS:CGROUP,DS:DGROUP
```

231422-58

```
     ;
     ; CHAR$OUT: PROCEDURE(CHAR) EXTERNAL;
     ; DECLARE CHAR BYTE;
     ; END CHAR$OUT;
     ; Outputs character to the screen.
     ; DOS system call 2
     ;
     CHAR                EQU      [BP+4]        STACK
     ;

     CHAROUT             PROC     NEAR          +-------+
                 PUBLIC  CHAROUT                ! CHAR  !  x
                 PUSH    BP                     +-------+
                 MOV     BP,SP                  !IP lo  !  x-1
                 MOV     DL,CHAR                +-------+
                 MOV     AH,2                   !IP hi  !  x-2
                 INT     DOS                    +-------+
                 POP     BP                     !BP lo  !  x-3
                 RET     2                      +-------+
     CHAROUT             ENDP                   !BP hi  !  x-4   <--SP
     ;                                          +-------+
     ; KEYIN$NOECHO: PROCEDURE BYTE EXTERNAL;
     ; END KEYIN$NOECHO;
     ; Reads character without echoing to display
     ;

     KEYINNOECHO PROC    NEAR
                 PUBLIC  KEYINNOECHO
                 MOV     AH,8                   (DOS call 8)
                 INT     DOS
                 RET
     KEYINNOECHO ENDP
```

Figure 7-56. I/O Routines for IBM/PC (continued)

231422-59

Figure 56. I/O Routines for IBM/PC

```
;
; MSG$OUT: PROCEDURE(MSG$PTR) EXTERNAL;
; DECLARE MSG$PTR POINTER;
; END MSG$OUT;
; /* NOTE: MESSAGE IS TERMINATED WITH A DOLLAR SIGN */
; MSG$PTR is double word pointer SEG:OFFSET
MSG_L               EQU     [BP+4]
MSG_H               EQU     [BP+6]
;
MSGOUT              PROC    NEAR
          PUBLIC    MSGOUT
;
          PUSH      BP
          MOV       BP,SP
          MOV       DX,MSG_L
          PUSH      DS
          MOV       AX,MSG_H
          MOV       DS,AX
          MOV       AH,9                (DOS call 9)
          INT       DOS
          POP       DS
          POP       BP
          RET       4
MSGOUT              ENDP

;
; LINE$IN: PROCEDURE(LINE$PTR) EXTERNAL;
; DECLARE LINE$PTR POINTER;
; END LINE$IN
;

LINE_L              EQU     [BP+4]
LINE_H              EQU     [BP+6]
;
LINEIN              PROC    NEAR
          PUBLIC    LINEIN
          PUSH      BP
          MOV       BP,SP
          PUSH      DS
          MOV       AX,LINE_H
          MOV       DS,AX
          MOV       DX,LINE_L
          MOV       AH,10                (DOS call 10)
          INT       DOS
          POP       DS
          POP       BP
          RET       4
LINEIN              ENDP
```

231422-60

```
;
; KEY$STAT: PROCEDURE BYTE EXTERNAL;
; END KEY$STAT;
; Indicates whether any keyboard key was pressed.

KEYSTAT             PROC    NEAR
          PUBLIC    KEYSTAT
          MOV       AH,11               (DOS call 11)
          INT       DOS
          RET
KEYSTAT             ENDP
;
;
CODE                ENDS
          END
```

231422-61

Figure 56. I/O Routines for IBM/PC (Continued)

7.2 Initialization and Declarations

Figure 57 shows some declarations describing what addresses the devices have and also some literals to help understand the other routines in this section.

Figure 58 shows the initialization routines for the IBM PC and for the 82588. It also shows some of the typical values taken by the memory buffers for Configure, IA_Set, Multicast and transmit buffers.

Following are some literal declarations that are used in the procedure examples

```
Following are some literal  declarations  that  are  used  in  the
procedure examples
  declare

   cs_588        literally '0300h' , /* 82588 COMMAND/STATUS          */
   brd_port            literally '0301h' , /* DMA/INTERUPT ENABLE PORT     */
   pic_mask            literally '021h'  , /* 8259A MASK REGISTER          */
   pic_ocw2            literally '020h'  , /* 8259A COMMAND WORD 2         */
   dma_mask            literally '0ah'   , /* 8237A MASK REGISTER          */
   dma_mode            literally '0bh'   , /* 8237A MODE REGISTER          */
   dma_flff            literally '0ch'   , /* 8237A 1ST/2ND BYTE FLOP      */
   dma_addr_1          literally '02h'   , /* 8237A CHANNEL 1 ADDR. REG.   */
   dma_bc_1            literally '03h'   , /* 8237A CHANNEL 1 BYTE COUNT   */
   dma_addrh_1         literally '083h'  , /* CHANNEL 1 PAGE REGISTER      */
   dma_addr_3          literally '06h'   , /* 8237A CHANNEL 3 ADDR. REG.   */
   dma_bc_3            literally '07h'   , /* 8237A CHANNEL 3 BYTE COUNT   */
   dma_addrh_3         literally '082h'  , /* CHANNEL 3 PAGE REGISTER      */
   dma_on_1            literally '01h'   , /* START CHANNEL 1              */
   dma_on_3            literally '03h'   , /* START CHANNEL 3              */
   dma_off_1           literally '05h'   , /* STOP CHANNEL  1              */
   dma_off_3           literally '07h'   , /* STOP CHANNEL  3              */
   enable_588          literally '0dfh'  , /* UNMASK INTERRUPT LEVEL 5     */
   seoi_pico           literally '065h'  , /* SPECIFIC EOI LEVEL 5         */
   tx_dir              literally   '1'   , /* MEMORY TO 82588              */
   rx_dir              literally   '0'   , /* 82588 TO MEMORY              */
   dma_rx_mode_1       literally '045h'  , /* RX ON CHANNEL # 1            */
   dma_rx_mode_3       literally '047h'  , /* RX ON CHANNEL # 3            */
   dma_tx_mode_1       literally '049h'  , /* TX ON CHANNEL # 1            */
   dma_tx_mode_3       literally '04bh'  , /* TX ON CHANNEL # 3            */
```

231422-62

Figure 57. Literal Declarations

Initialization Routines

```
Initialization routines

/* SYSTEM INITIALIZE */

sys_init: procedure;

   call set$interrupt (13,intr_588);                    /* BASE 8, LEVEL 5      */
   output(pic_mask) = input(pic_mask) and enable_588;  /* ENABLE 588 INTERR. */
   output(pic_ocw2) =     seoi_pico;                   /* ACKS PENDING INTERR*/

   wr_ptr,rd_ptr,fifocnt=0;                            /* RESET STATUS FIFO  */

   /***************************************************/
   /* CONVERT SEG:OFFSET FORMAT TO 20 BIT ADDRESSES   */
   /* FOR ALL THE BUFFERS                             */
   /***************************************************/

   iaset_dma_addr = convert_20bit_addr(@ia_set_buff_588(0));
   cnf_dma_addr   = convert_20bit_addr(@config_588(0));
   dmp_dma_addr   = convert_20bit_addr(@dump_buff_588(0));
   mc_dma_addr    = convert_20bit_addr(@multicast_buff_588(0));
   tx_dma_addr    = convert_20bit_addr(@tx_buffer_588(0));
   do i=0 to 7 ;
     rx_dma_addr(i)=convert_20bit_addr(@rx_buffer(i).buff(0));
   end;

   output(brd_port)=0ffh;  /* ENABLE DMA AND INTERRUPT DRIVERS */

   end sys_init;

82588 initialization

init_588: procedure;

  config_588(00) = 10;          /* TO CONFIGURE ALL 10 PARAMETERS          */
  config_588(01) = 00;
  config_588(02  = 00001000b;   /* MODE 0, 8  MHZ CLOCK, 1 MB/S            */
  config_588(03) = buff_len/4;  /* RECEIVE BUFFER LENGTH                   */
  config_588(04) = 00100110b;   /* NO LOOPBACK, ADDR LEN = 6, PREAMBLE = 8 */
  config_588(05) = 00000000b;   /* DIFFERENTIAL MANCHESTER = OFF           */
  config_588(06) = 96;          /* IFS = 96 TCLK                           */
  config_588(07) = 0;           /* SLOT TIME = 512 TCLK                    */
  config_588(08) = 11110010b;   /* MAX. NO. RETRIES = 15                   */
  config_588(09) = 00000100b;   /* MANCHESTER ENCODING                     */
  config_588(10) = 10001000b;   /* INTERNAL CRS AND CDT, CRSF = 0          */
  config_588(11) = 64;          /* MIN FRAME LENGTH = 64 BYTES = 512 BITS  */
```

231422-63

Figure 58. Initialization Routines

```
ia_set_buff_588(0) = 6;
ia_set_buff_588(1) = 0;
ia_set_buff_588(2) = 000h;
ia_set_buff_588(3) = 041h;
ia_set_buff_588(4) = 000h;
ia_set_buff_588(5) = 000h;
ia_set_uff_588(6)  = 000h;
ia_set_buff_588(7) = 000h;

multicast_buff_588(00) = 12;
multicast_buff_588(01) = 00h;
multicast_buff_588(02) = 11h;
multicast_buff_588(03) = 12h;
multicast_buff_588(04) = 13h;
multicast_buff_588(05) = 14h;
multicast_buff_588(06) = 15h;
multicast_buff_588(07) = 16h;
multicast_buff_588(08) = 21h;
multicast_buff_588(09) = 22h;
multicast_buff_588(10) = 23h;
multicast_buff_588(11) =24h;
multicast_buff_588(12) = 25h;
multicast_buff_588(13) = 26h;

tx_buffer_588(00) = tx_frame_len mod 256;
tx_buffer_588(01) = tx_frame_len / 256;
tx_buffer_588(02) = 011h;    /* INITIAL DESTINATION ADDRESS = MC(1) */
tx_buffer_588(03) = 012h;
tx_buffer_588(04) = 013h;
tx_buffer_588(05) = 014h;
tx_buffer_588(06) = 015h;
tx_buffer_588(07) = 016h;

end init_588;
```

231422-64

Figure 58. Initialization Routines (Continued)

7.3 General Commands

Operations like Transmit, Receive, Configure, etc. are done by a simple sequence of loading the DMA controller with the necessary parameters and then writing the command to the 82588.

Example: Configure Command

To configure the operating environment of the 82588. This command must be the first one to be executed after a RESET.

```
call
DMA_LOAD(1,1,12,@CONFIG_588_ADDR);
output (CS_588) = 12h;
```

The first statement is the prologue to the configure command to the 82588 which calls a routine to load and initialize the DMA controller for the desired operation. This routine is described in section 7.4. The parameters for DMA__LOAD are:

```
first parameter  = 82588 channel
                   number ( = 1)
second parameter = direction ( = 1,
                   memory >> 82588)
third parameter  = length of DMA
                   transfer ( = 12)
fourth parameter = pointer to a 20 bit
                   address of the
                   memory buffer
                   (=@CONFIG_588_ADDR)
```

The second statement writes 12h to the command register of the 82588 to execute a Configure command on channel 1.

When the command execution is complete (successfully or not), 82588 interrupts the 8088 CPU through the 8259A, on the system board. This executes the interrupt service routine, described in section 7.5, which takes the epilogue action for the command.

Most operations are very similar in structure to Configure. The 82588 Reference Manual describes them in detail. Figure 59 shows a listing of the most commonly used operations like:

```
CONFIGURE     INDIVIDUAL-ADDRESS (IA)
              SET-UP
TRANSMIT      MULTICAST-ADDRESS (MC)
              SET-UP
DIAGNOSE      RECEIVE (RCV)-ENABLE
DUMP          RECEIVE (RCV)-DISABLE
TDR           RECEIVE (RCV)-STOP
RETRANSMIT    READ-STATUS
```

```
ia_set: procedure public;                    /* COMMAND - 01 */

    call dma_load(cmd_channel,tx_dir,8,@iaset_dma_addr);

    /* SET DMA CHANNEL 0 OR 1 TO TRANSFER FROM MEMORY
       TO THE 82588. iaset_dma_addr VARIABLE STORES THE
       20 BIT POINTER TO THE INDIVIDUAL ADDRESS BUFFER   */

    if cmd_channel then output (cs_588) = 11h;
    else output(cs_588) = 01h;

    /* EVERY COMMAND CAN BE EXECUTED IN EITHER DMA CHANNEL 0 OR 1.
       THE VARIABLE cmd_channel INDICATES THE REQUIRED CHANNEL  */

end ia_set;

/*---------------------------------------------------------------------------*/

config: procedure public;                    /* COMMAND - 02 */

    call dma_load(cmd_channel,tx_dir,12,@cnf_dma_addr);
    if cmd_channel then output (cs_588) = 12h;
    else output(cs_588) = 02h;

end config;

/*---------------------------------------------------------------------------*/

multicast: procedure public;                 /* COMMAND - 03 */

    call dma_load(cmd_channel,tx_dir,14,@mc_dma_addr);
    if cmd_channel then output (cs_588) = 13h;
    else output(cs_588) = 03h;

end multicast;

/*---------------------------------------------------------------------------*/

transmit: procedure(buffer_len) public;      /* COMMAND - 04 */

     declare buffer_len word;

    tx_buffer_588(00) = low(buffer_len);
    tx_buffer_588(01) = high(buffer_len);

    call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);

    if cmd_channel then output (cs_588) = 14h;
    else output(cs_588) = 04h;

end transmit;
```

231422-65

Figure 59. General Commands

```
tdr: procedure public;                          /* COMMAND - 05 */

    if cmd_channel then output (cs_588) = 15h;
    else output(cs_588) = 05h;

end tdr;

/*--------------------------------------------------------------------------*/

dump_588: procedure public;                     /* COMMAND - 06 */

    call dma_load(cmd_channel,rx_dir,64,@dmp_dma_addr);
    if cmd_channel then output (cs_588) = 16h;
    else output(cs_588) = 06h;

end dump_588;

/*--------------------------------------------------------------------------*/

diagnose: procedure public;                     /* COMMAND - 07 */

    if cmd_channel then output (cs_588) = 17h;
    else output(cs_588) = 07h;

end diagnose;

/*--------------------------------------------------------------------------*/

rcv_enable: procedure(channel,buffer_no,len) public;    /* COMMAND - 08 */

    declare channel byte;
    declare len  word;
    declare buffer_no byte;
    call dma_load(channel,rx_dir,len,@rx_dma_addr(buffer_no));
    if rx_channel then output (cs_588) = 18h;
    else output(cs_588) = 08h;

end rcv_enable;

/*--------------------------------------------------------------------------*/

rcv_disable: procedure public;                  /* COMMAND - 10 */

    enable_rcv=0;
    output(cs_588)=0ah;

end rcv_disable;
```

231422-66

```
/*--------------------------------------------------------------------------*/

rcv_stop: procedure public;                     /* COMMAND - 11 */

    enable_rcv=0;
    output(cs_588)= 0bh;

end rcv_stop;

/*--------------------------------------------------------------------------*/

retransmit: procedure public;                   /* COMMAND - 12 */

    call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);
    if cmd_channel then output (cs_588) = 1ch;
    else output(cs_588) = 0ch;

end retransmit;

/*--------------------------------------------------------------------------*/

abort: procedure public;                        /* COMMAND - 13 */

    output(cs_588)= 1dh;
    call new_status(1);

end abort;

/*--------------------------------------------------------------------------*/

reset_588: procedure public;                    /* COMMAND - 14 */

    enable_rcv=0;
    output(cs_588) = 1eh;
    call config;

end reset_588;
```

231422-67

Figure 59. General Commands (Continued)

7.4 DMA Routines

DMA_LOAD procedure is used to program the 8237A DMA controller for all the operations requiring DMA service. It also starts or enables the programmed DMA channel after programming it. Figure 60 shows the listing of this procedure. It accepts 4 parameters from the calling routine to decide the programming configuration for the 8237A. The parameters for DMA_LOAD are: Channel, direction, buff_len, and buff_addr.

```
Converting a pointer SEG:OFFSET to a 20 bit address
convert_20bit_addr:procedure(ptr) dword public;

   declare   ptr        pointer,
             ptr_addr   pointer,
             ptr_20bit  dword   ,
             (wrd   based ptr_addr)(2) word;

   ptr_addr=@ptr;
   ptr_20bit=shl((ptr_20bit:=wrd(1)),4)+wrd(0);
   return(ptr_20bit);

end convert_20bit_addr;


IBM/PC DMA loading procedure

dma_load: procedure(channel,direction,buff_len,buff_addr) reentrant public;

    declare channel byte;              /* CHANNEL #, 0 or 1                   */
    declare direction byte;            /* 0=RX, 588 -> MEM; 1=TX, MEM -> 588 */
    declare buff_len word;             /* BYTE COUNT                          */
    declare buff_addr pointer;         /* BUFFER ADDR IN 20 BITS FORM         */
    declare (wrd based buff_addr)(2) word;

    channel=channel and 1;             /* GET LEAST SIGNIFICANT BIT           */

    if channel=0 then                  /* EXECUTE COMMAND ON CHANNEL 1        */
      do;
      output(dma_flff)  = 0;           /* CLEAR FIRST/LAST FLIP-FLOP          */
      if direction=0
          then output(dma_mode)=dma_rx_mode_1;  /* DIRECTION BIT, TELLS      */
      else output(dma_mode)=dma_tx_mode_1;      /* TRANSMIT OR RECEIVE       */
      output(dma_addr_1)  = low (wrd(0));       /* LOAD LSB ADDRESS BYTE     */
      output(dma_addr_1)  = high(wrd(0));       /* LOAD MSB ADDRESS BYTE     */
      output(dma_addrh_1) = low (wrd(1));       /* LOAD PAGE REGISTER        */
      output(dma_bc_1)    = low (buff_len);     /* LOAD LSB BYTE COUNT       */
      output(dma_bc_1)    = high(buff_len);     /* LOAD MSB BYTE COUNT       */
      output(dma_mask) = dma_on_1;              /* START CHANNEL 1           */
      end;
    else do;                          /* SAME AS BEFORE FOR CHANNEL 3        */
      output(dma_flff)  = 0;
      if direction=0
          then output(dma_mode)=dma_rx_mode_3;
      else output(dma_mode)=dma_tx_mode_3;
      output(dma_addr_3)  = low (wrd(0));
      output(dma_addr_3)  = high(wrd(0));
      output(dma_addrh_3) = low (wrd(1));
      output(dma_bc_3)    = low (buff_len);
      output(dma_bc_3)    = high(buff_len);
      output(dma_mask) = dma_on_3;
      end;

end dma_load;
```

231422-68

Figure 60. DMA Routine

One peculiarity about this procedure is that in order to speed up the DMA step-up, this procedure doesn't get a pointer to the buffer, but a pointer to a 20 bit address in the 8237 format. The 8088/8086 architecture define pointers as 32 bits seg:offset entities, where seg and offset are 16 bit operands. By the other hand the IBM/PC uses an 8237A and a page register, requiring a memory address to be a 20 bit entity. The process of converting a seg:offset pointer to a 20 bit address is time consuming and could negatively affect the performance of the 82588 driver software. The decision was to make the pointer/address conversions during initialization, considering that the buffers are static in memory (essentially removing this calculation from the real time response loops).

Figure 61 is a listing of the DMA_LOAD procedure for the 80188 or 80188 on-chip DMA controller. It has the same caller interface as the 8237A based one.

```
dma_load: procedure(channel,direction,trans_len,buff_addr) reentrant;

/* To load and start the  80186 DMA controller for the desired operation */

   declare dma_rx_mode   literally '1010001001000000b'; /* rx channel */
         /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

   declare dma_tx_mode   literally '0000110100000000b'; /* tx channel */
         /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */


   declare channel byte;        /* channel #                               */
   declare direction byte;      /* 0 = rx, 588 -> mem; 1 = tx, mem -> 588 */
   declare trans_len word;      /* byte count                              */
   declare buff_addr pointer; /* buffer pointer in 20 bit addr. form      */

   declare (wrd based buff_addr)(2) word;

   do case channel and 00000001b;
     do case direction and 00000001b;
       do;             /* channel 0, 588 to memory  */
       output(dma_0_dpl)  = wrd(0);
       output(dma_0_dph)  = wrd(1);
       output(dma_0_spl)  = ch_a_588;
       output(dma_0_sph)  = 0;
       output(dma_0_tc)   = trans_len;
       output(dma_0_cw)   = dma_rx_mode or 0006h; /* Start DMA chl 0 */
       end;

       do;             /* channel 0, memory to 588 */
       output(dma_0_dpl)  = ch_a_588;
       output(dma_0_dph)  = 0;
       output(dma_0_spl)  = wrd(0);
       output(dma_0_sph)  = wrd(1);
       output(dma_0_tc)   = trans_len;
       output(dma_0_cw)   = dma_tx_mode or 0006h; /* Start DMA chl 0 */
       end;
     end;
```

231422-69

Figure 61. 80186 DMA Routines

```
        do case direction and 00000001b;
           do;                 /* channel 1, 588 to memory  */
           output(dma_1_dpl)   = wrd(0);
           output(dma_1_dph)   = wrd(1);
           output(dma_1_spl)   = ch_b_588;
           output(dma_1_sph)   = 0;
           output(dma_1_tc)    = trans_len;
           output(dma_1_cw)    = dma_rx_mode or 0006h; /* Start DMA chl 1 */
           end;

           do;                 /* channel 1, memory to 588 */
           output(dma_1_dpl)   = ch_b_588;
           output(dma_1_dph)   = 0;
           output(dma_1_spl)   = wrd(0);
           output(dma_1_sph)   = wrd(1);
           output(dma_1_tc)    = trans_len;
           output(dma_1_cw)    = dma_tx_mode or 0006h; /* Start DMA chl 1 */
           end;
         end;

      end;

    end dma_load;
```

231422-70

Figure 61. 80186 DMA Routines (Continued)

7.5 Interrupt Routine

The interrupt service routine, 'intr__588', shown in Figure 62, is invoked whenever the 82588 interrupts. The main difficulty in designing this interrupt routine was to speed its performance. Fast status processing was a basic requirement to be able to handle back to back frames.

The interrupt handler will read 82588 status, and put them into a 64 byte long EVENT__FIFO. Those statuses are going to be used in the main loop for updating screen counters. All the statistics are updated as fast as possible in the interrupt handler to fulfill the back-to-back frame processing requirement.

The interrupt handler is not reentrant, interrupts are disabled at the beginning and reenabled on exit.

```
Interrupt service routine

intr_588:procedure interrupt 13;

     declare stat                          byte,
             event                         byte,
             i                             byte,
             (st0,st1,st2,st3)             byte,
             rx_st0                        byte,
             rx_st1                        byte;

    /* FOLLOWING LITERALS HAVE THE PURPOSE OF ENABLE ACTING
       ON EITHER CHANNEL 1 OR 3 SELECTIVELY                        */

    declare

       stop_cmd_dma   literally 'if cmd_channel
                                     then output(dma_mask)=dma_off_3;
                                   else output(dma_mask)=dma_off_1',
       stop_rx_dma    literally 'if rx_channel
                                     then output(dma_mask)=dma_off_3;
                                   else output(dma_mask)=dma_off_1',

       issue_rtx_cmd literally 'if cmd_channel
                                     then output(cs_588)=1Ch;
                                   else output(cs_588)=0ch',
       issue_tx_cmd   literally 'if cmd_channel
                                     then output(cs_588)=14h;
                                   else output(cs_588)=04h';

     disable;                                    /* DISABLE INTERRUPTS    */
                                                 /* NO INTERR. NESTING    */
     output(cs_588) =0fh;                        /* RLS 588 PTR, START 0 */

     event_fifo(wr_ptr).st0,st0=input(cs_588);  /* READ 82588 STATUS     */
     event_fifo(wr_ptr).st1,st1=input(cs_588);  /* REGISTERS, PASSING    */
     event_fifo(wr_ptr).st2,st2=input(cs_588);  /* THEM TO THE MAIN      */
     event_fifo(wr_ptr).st3,st3=input(cs_588);  /* PROGRAM ON THE FIFO   */

     wr_ptr=(wr_ptr+1) and 0fh;                 /* INCREMENT FIFO        */
     fifocnt=(fifocnt+1) and 0fh;               /* COUNTERS              */

     event=st0 and 0fh;                         /* GET EVENT FIELD       */

     output(cs_588)=80h;                        /* ACKNOWLEDGE 82588     */
                                                /* INTERRUPT             */
```

231422-71

```
   do case event;

   ev_00: ;                                /* NOP COMMAND           */
   ev_01: stop_cmd_dma;                    /* IA_SETUP, STOP DMA    */
   ev_02: stop_cmd_dma;                    /* CONFIGURE, STOP DMA   */
   ev_03: stop_cmd_dma;                    /* MULTICAST, STOP DMA   */
   ev_04: do;                              /* TRANSMIT DONE         */
          stop_cmd_dma;

        /* CHECK IF THERE WAS A COLLISION AND IS NOT THE
           MAX COLLISION                                            */

        stat=(st2 and 10000000b) or (st1 and 00100000b);
        if (stat=80h)
           then do;                 /* RETRANSMIT                   */
                call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr);
                issue_rtx_cmd;
                /* UPDATE STATISTICS                                */
                total_tx_count=total_tx_count+1;
                coll_cnt(17) = coll_cnt(17) + 1; /*TOTAL COLL*/
                bad_tx_count = bad_tx_count + 1;
                end;
        else do;
             if in_loop     /* EXECUTING TRANSMISSIONS IN LOOP  */
              then do;      /* RE ISSUE TRANSMIT COMMAND        */
                   call dma_load(cmd_channel,tx_dir,1536,@tx_dma_addr
                   issue_tx_cmd;
                   total_tx_count=total_tx_count+1;
                   end;
             if (st2 and 00100000b) = 0          /* BAD TRANSMIT*/
               then do;
                    bad_tx_count = bad_tx_count + 1;
                    /* INCREMENT UNDERRUN COUNTER                   */
                    tmp=scr(tmp:=st2,1);
                    tx_under=tx_under plus 0;
                    /* INCREMENT LOST CTS COUNTER                   */
                    tmp=scr(tmp,1);
                    lost_cts=lost_cts plus 0;
                    /* INCREMENT LOST CRS COUNTER                   */
                    tmp=scr(tmp,1);
                    lost_crs=lost_crs plus 0;
                    if (stat=0A0h)  /* INC COLLISIONS COUNTER   */
                        then coll_cnt(17) = coll_cnt(17) + 1;
                    end;
             end;

             /* INCREMENT DEFER  COUNTER                            */
             tmp=scl((tmp:=st1),1);
             tx_defer=tx_defer plus 0;
        end;
```

231422-72

Figure 62. Interrupt Routine

```
ev_05: stop_cmd_dma;                  /* TDR COMMAND, STOP DMA     */
ev_06: stop_cmd_dma;                  /* DUMP COMMAND, STOP DMA    */
ev_07: stop_cmd_dma;                  /* DIAGNOSE CMD, STOP DMA    */
ev_08:                                /* RECEIVED FRAME            */
    do;
    stop_rx_dma;
    i=(current_buff+1) and 00000111b; /* INC BUFFER NO. MOD 8*/
    if enable_rcv<>0                  /* IF RECEIVER IS ON     */
        then do;                      /* PREPARE NEXT BUFFER */
             call dma_load(rx_channel,rx_dir,1532,@rx_dma_addr(i));
             if rx_channel then output(cs_588)= 18h;
             else output(cs_588)=08h;
             rx_buffer(i).chain_cnt=0;
             end;
    else call rcv_disable;            /* DISABLE RECEIVER      */

    /* FIND ADDRESS OF END OF CURRENTLY RECEIVED BUFFER       */
    /* BY CALCULATING IT WITH THE 82588 BYTE COUNT REGS.      */
    rx_buff_off=(shl(double(st2),8) or double(st1));
    /* READ STATUS BYTES FROM MEMORY                           */
    rx_st0=rx_buffer(current_buff).buff(rx_buff_off-2);
    rx_st1=rx_buffer(current_buff).buff(rx_buff_off-1);
    /* UPDATE ACTUAL BUFFER SIZE                               */
    rx_buffer(current_buff).actual_size=rx_buff_off;
    rx_buffer(current_buff).st0=rx_st0;
    rx_buffer(current_buff).st1=rx_st1;
    current_buff=i;
    /* UPDATE TOTAL RECEIVED BUFFERS                           */
    total_rcv_count=total_rcv_count+1;
    /* UPDATE STATISTICS                                       */
    if (rx_st1 and 00100000b)=0
        then do;
             bad_rcv_count=bad_rcv_count+1;
             /* INCREMENT  NO END OF FRAME COUNTER            */
             tmp=scr(tmp:=rx_st0,7);
             no_eof=no_eof plus 0;
             /* INCREMENT  SHORT FRAME COUNTER                */
             tmp=scr(tmp,1);
             srt_frm=srt_frm plus 0;
             /* INCREMENT  RX OVERRUN COUNTER                 */
             tmp=scr(tmp:=rx_st1,1);
             rx_over=rx_over plus 0;
             /* INCREMENT  ALIGNMENT ERROR COUNTER            */
             tmp=scr(tmp,2);
             alg_err=alg_err plus 0;
             /* INCREMENT  CRC ERROR COUNTER                  */
             tmp=scr(tmp,1);
             crc_err=crc_err plus 0;
             end;
    end;
```

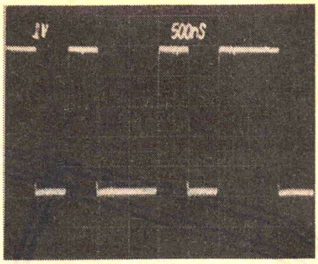
231422-73

```
            /*  EV_09 REQUESTS ASSIGNMENT OF A NEW BUFFER        */
    ev_09:  call allocate_new_buffer(not(rol(st3,1)) and 00000001b);
    ev_10:  stop_rx_dma;       /* RECEIVE DISABLE                 */
    ev_11:  stop_rx_dma;       /* STOP RECEIVE                    */
    ev_12:  do;                /* RE-TRANSMIT DONE                */
            stat=(st2 and 10000000b) or (st1 and 00100000b);
            if (stat=80h)
            then do;           /* RETRANSMIT                      */
                 call dma_load(1,tx_dir,1536,@tx_dma_addr);
                 issue_rtx_cmd;
                 coll_cnt(17) = coll_cnt(17) + 1;
                 total_tx_count=total_tx_count+1;
                 bad_tx_count=bad_tx_count +1;
                 end;
            else do;
                 if in_loop
                  then do;    /* LOOP RETRANSMISSIONS            */
                        call dma_load(cmd_channel,tx_dir,1536,@tx_dma_a
                        issue_tx_cmd;
                        total_tx_count=total_tx_count+1;
                        end;
                 if (stat=0A0h) /* MAX COLLISION                  */
                    then do;
                          coll_cnt(16) = coll_cnt(16)+1;
                          coll_cnt(17) = coll_cnt(17)+1;
                          bad_tx_count=bad_tx_count +1;
                          end;
                 /* UPDATE SPECIFIC COLLISION COUNTER             */
                 else  coll_cnt(st1 and 0fh)
                             = coll_cnt(st1 and 0fh) + 1;
                 end;
            end;
    ev_13:  stop_cmd_dma;       /* EXECUTION ABORTED              */
    ev_14:  ;
    ev_15: stop_cmd_dma;        /* DIAGNOSE FAILED                */
    end;

  /*  ACKNOWLEDGE 8259A  INTERRUPT                                */
  output(pic_ocw2)= seoi_pico;  /* SPECIFIC EOI FOR 8259          */

end intr_588;
```

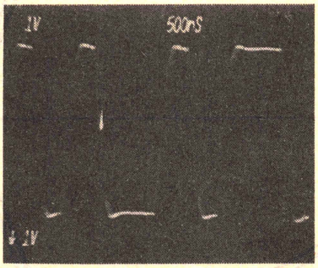
231422-74

Figure 62. Interrupt Routine (Continued)

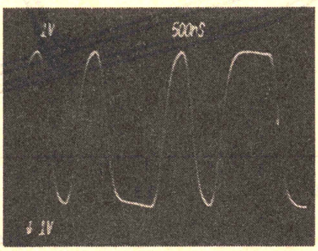
APPENDIX A
STARLAN SIGNALS



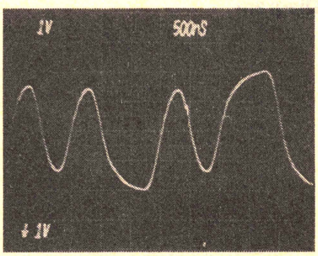
231422-51



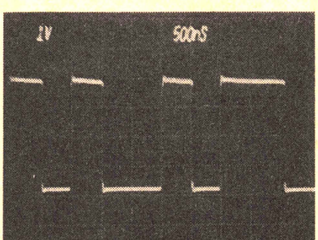
231422-52



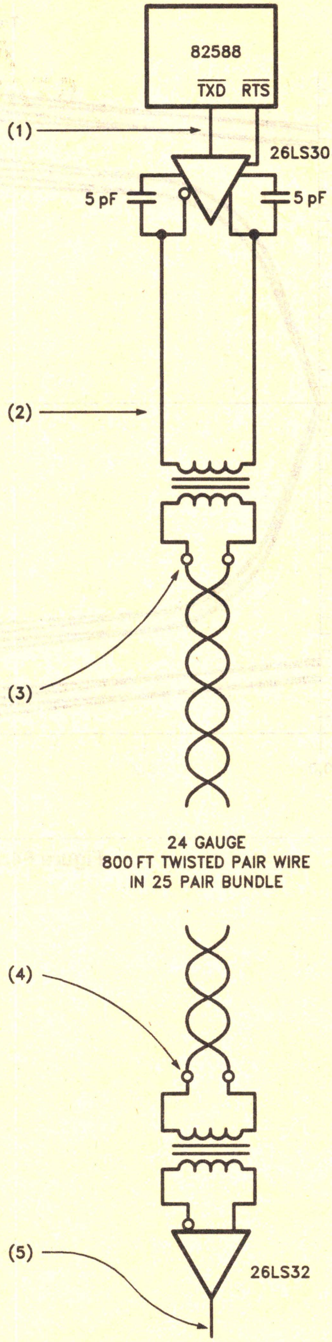
231422-53



231422-54



231422-55



231422-47

Figure 63. StarLAN Signals

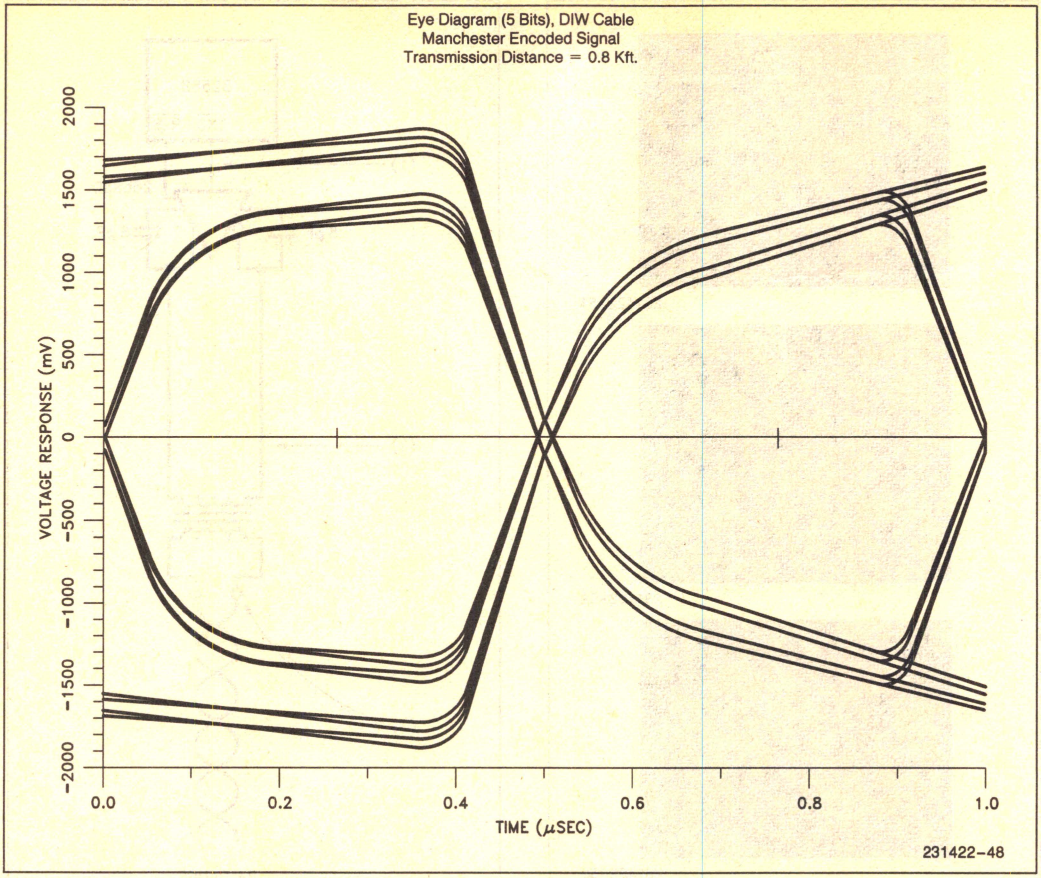


Figure 64. Received Signal Eye Diagram

APPENDIX B
802.3 1BASE5 MULTI-POINT EXTENSION (MPE)

As previously stated, one of the most important advantages of StarLAN is being able to work on already installed phone wires. This advantage is considerably diminished in Europe where numerous constraints exist to the using of those wires:

1. Wire belongs to local PTTs.
2. Not enough spare wires.

This same issue is raised when talking about small businesses where in a lot of cases no wiring closets and/or spare wires are available.

In summary, in a lot of cases rewiring will be necessary, in which case the STAR topology may not be the most economical one.

Recently the StarLAN 802.3 1BASE5 task force has been considering the extension of the StarLAN base topology. This extension called MULTI POINT EXTENSION (MPE) is going to be developed to address the previously described marketing requirements.

Currently no agreement has been reached by the StarLAN task force on the MPE exact topology and implementation. Multiple approaches have been presented, but no consensus met. It was decided though that the MPE is going to be an addendum to the STAR topology, and that its final specification will happen after the approval of the current 1BASE5 STAR topology (July 1986).

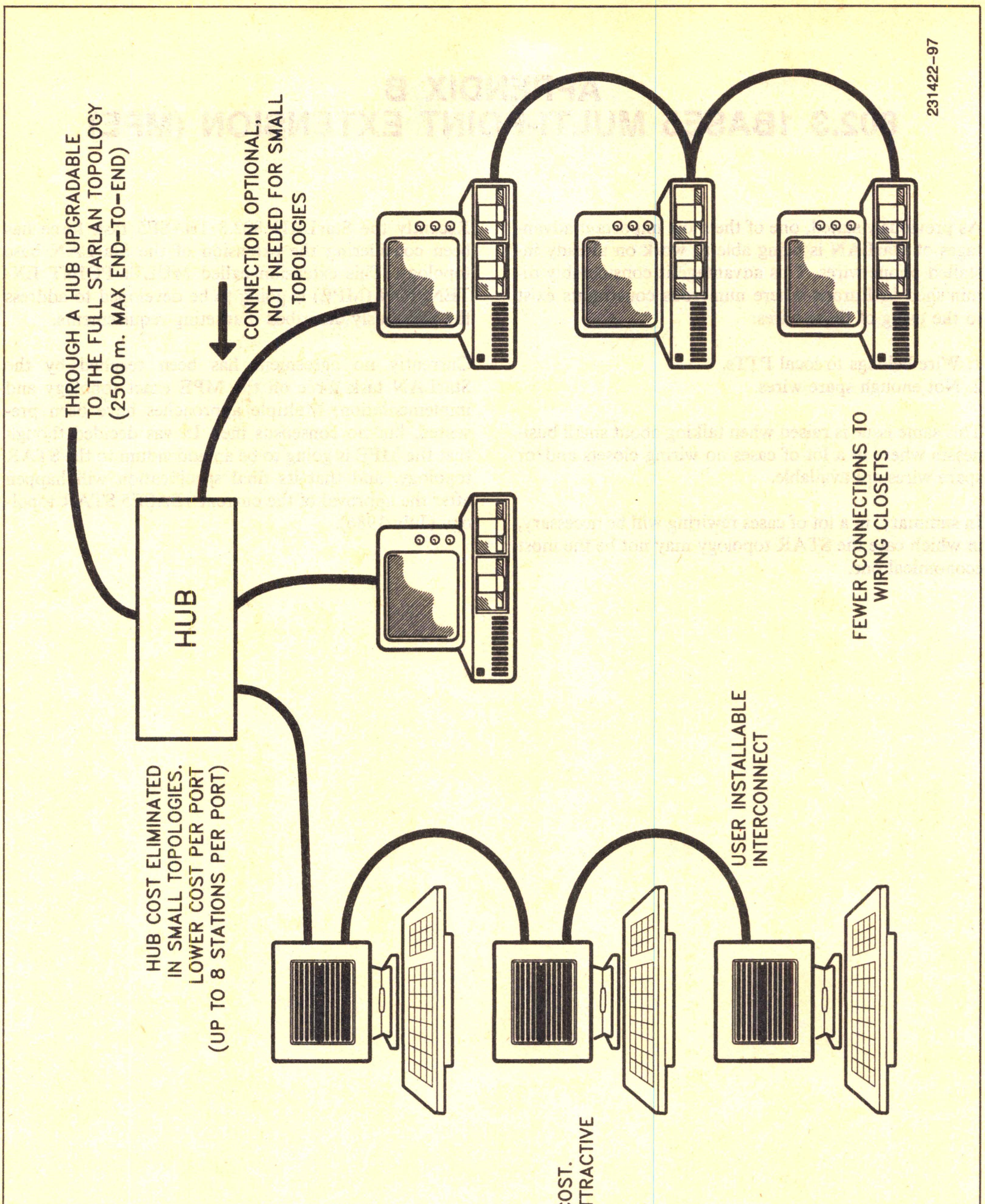


Figure 65. Multipoint Extension

APPENDIX C
SINGLE DMA CHANNEL INTERFACE

In a typical system, the 82588 needs 2 DMA channels to operate in a manner that no received frames are lost as discussed in section 5.1.3. If an existing system has only one DMA channel available, it is still possible to operate the 82588 in a way that no frames are lost. This method is recommended only in situations where a second DMA channel is impossible to get.

Figure 66 shows how the 82588 DMA logic is interfaced to one channel of a DMA controller. Two DRQ lines are ORed and go to the DMA controller DRQ line and the DACK line from the DMA controller is connected to DACK0 and DACK1 of the 82588. The 82588 is configured for multiple buffer reception (chaining), although the entire frame is received in a single buffer. Let us assume that channel CH-0 is used as the first channel for reception. After the ENAble RECeive command, CH-0 is dedicated to reception. As long as no frame is received, the other channel, CH-1, can be used for executing any commands like transmit, multicast address, dump, etc., by programming the DMA channel for the execution command. The status register should be checked for any ongoing reception, to avoid issuing an execution command when reception is active.

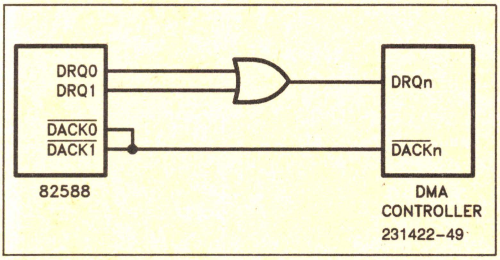


Figure 66. 82588 Using One DMA Channel

If a frame is received, an interrupt for additional buffer occurs immediately after an address match is established, as shown in Figure 67. After this, the received bytes start filling up the on-chip FIFO. The 82588 activates the DRQ line after 15—FIFO LIMIT + 3 bytes are ready for transfer in the FIFO (about 80 microseconds after the interrupt). The CPU should react to the interrupt within 80 μs and disable the DMA controller. It should also issue an ASSIGN ALTERNATE BUFFER command with INTACK to abort any execution command that may be active. The FIFO fills up in about 160 μs after interrupt. To prevent an underrun, the CPU must reprogram the DMA controller for frame reception and re-enable the DMA controller within 160 μs after the interrupt (time to receive about 21 bytes). No buffer switching actually takes place, although the 82588 generates request for alternate buffer every time it has no additional buffer. The CPU must respond to these interrupts with an ASSIGN ALTERNATE BUFFER command with INTACK. To keep the CPU overhead to a minimum, the buffer size must be configured to the maximum value of 1 kbyte.

If a frame transmission starts deferring due to the reception occurring just prior to an issued transmit command, the transmission can start once the link is free after reception. A maximum of 19 bytes are transmitted (stored in the FIFO and internal registers) followed by a jam pattern and then an execution aborted interrupt occurs. The aborted frame can be transmitted again.

If the transmit command is issued and the 82588 starts transmitting just prior to receiving a frame then transmit wins over receive—but this will obviously lead to a collision.

Note that the interrupt for additional buffer is used to abort an ongoing execution command and to program the DMA channel for reception just when a frame is received. This scheme imposes real time interrupt handling requirements on the CPU and is recommended only when a second DMA channel is not available.

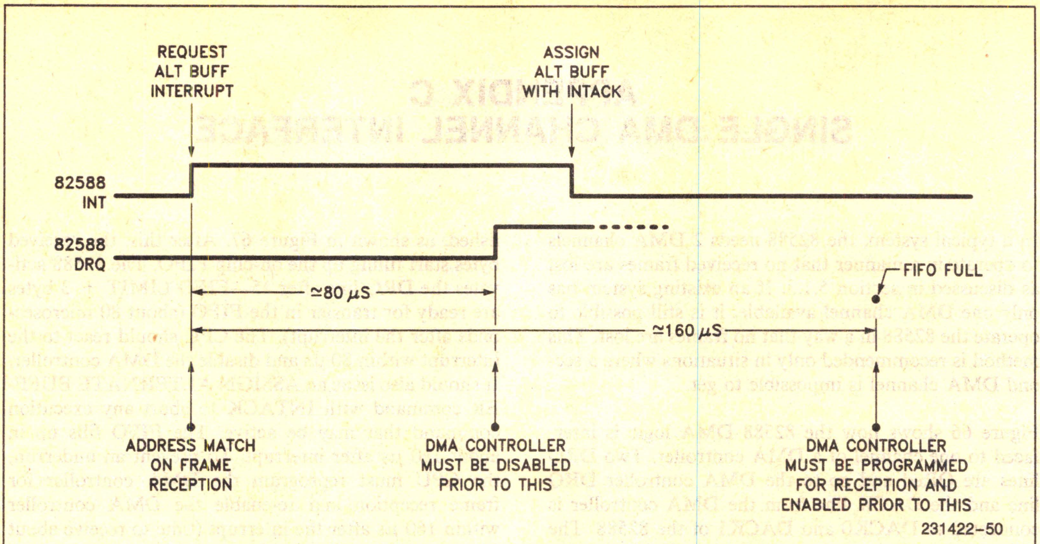


Figure 67. Timing at the Beginning of Frame Reception for Single DMA Channel Operation

APPENDIX D
MEASURING NETWORK DELAYS WITH THE 82588

Knowing networks round-trip delays in local area networks is an important capability. The round-trip delay very much defines the slot time parameter which by itself has a direct relationship to network efficiency and throughput. Very often the slot-time parameter is not flexible, due to standards requirements. Whenever it is flexible, optimization of this number may lead to significant improvement in network performance.

Another possible usage of the network delay knowledge is in balancing the inter-frame -spacing (IFS) on broadband networks. On those networks, stations nearer to the HEAD-END hear themselves faster than farther ones. Effectively having a shorter IFS than stations far from the HEAD-END. This difference causes an inbalance in network access time for different stations at different distances from the HEAD-END. Knowing the STATION/HEAD-END delay allows the user to reprogram the 82588 IFS accordingly, and by that balance the effective IFS for all the stations.

The 82588 has an internal mechanism that allows the user to measure this delay in BIT-TIME units. The method is based on the fact that the 82588 when configured for internal collision detection, requires that the carrier sense be active within half a slot-time after transmission has started. If this requirement is not fulfilled the 82588 notifies that a collision has occurred. Thus it is possible to configure the 82588 to different slot time values, then transmit a long frame (of at least half a slot-time). If the transmission succeeds, the network round-trip delay is less than half the programmed slot-time. If a collision is reported, the delay is longer. The value of the round-trip delay can be found by repeating this experiment process while scanning the slot-time configuration parameter value and searching the threshold. A binary search algorithm is used for that purpose. First the slot-time is configured for the maximum (2048 bits) and according if there was a collision or not, the number changed for the next try. (See Figure 68)

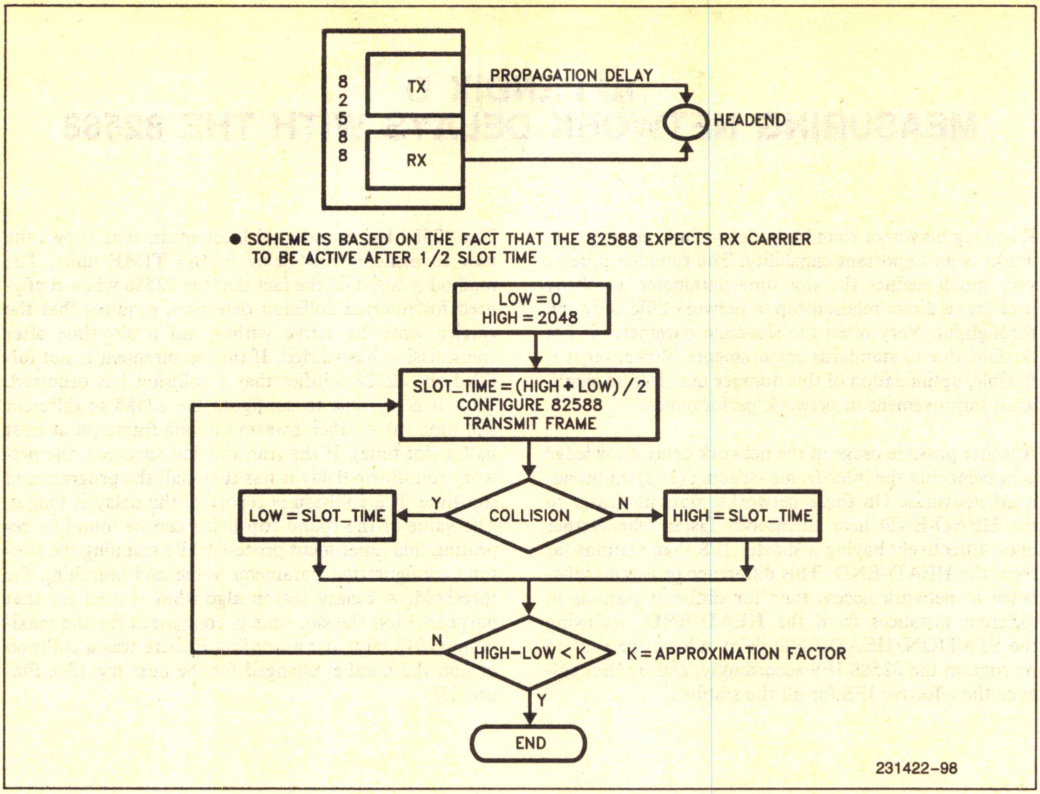


Figure 68. Network Delay Measurement using the 82588

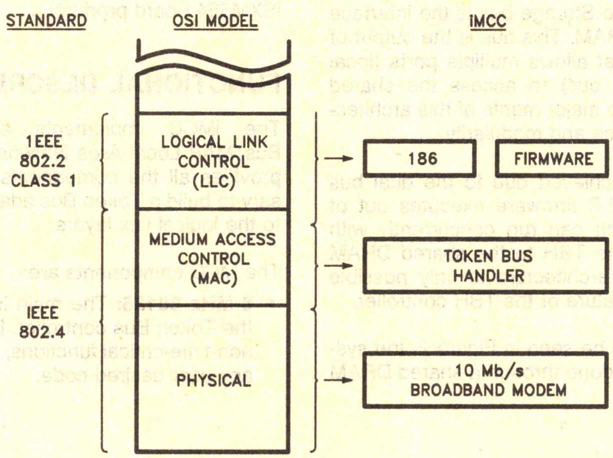
iMCC
MAP COMUNICATIONS CONTROLLER COMPONENT SET

OVERVIEW

Essential building blocks for implementing IEEE 802.4 Token Bus/Manufacturing Automation Protocol (MAP) 10-Mb/s broadband communication nodes. A full implementation of the 802.4 Media Access Control (MAC) and IEEE 802.2 Class I Logical Link Control (LLC) functions is supplied, plus ample processing power for higher level protocols. A completely assembled and tested circuit board provides the broadband modem function. An adapter interfacing any computer bus can be quickly implemented with the design aids provided. Component set as follows:

- **Token Bus Handler. Consists of Two CMOS VLSI Components Implementing Time-Critical MAC Functions**
- **10-Mb/s Modem Circuit Board. Available in General Purpose or IBM PC Format**
- **Firmware Implementing 802.2 LLC and Non-Time-Critical 802.4 MAC Functions. Firmware is Supplied in Two Intel 27128 EPROMs**
- **8-MHz 80186 Microprocessor**

iMCC features a high performance dual bus architecture for concurrent data flow and protocol execution; a well defined high level software interface; and a hardware interface which enables interfacing the basic controller to any computer bus. Implementation of the LLC and a subset of MAC in firmware allows easy migration as protocols evolve.



231929–1

Portions of material reprinted by permission of Industrial Networking Inc.

December 1986
Order Number: 231929-001

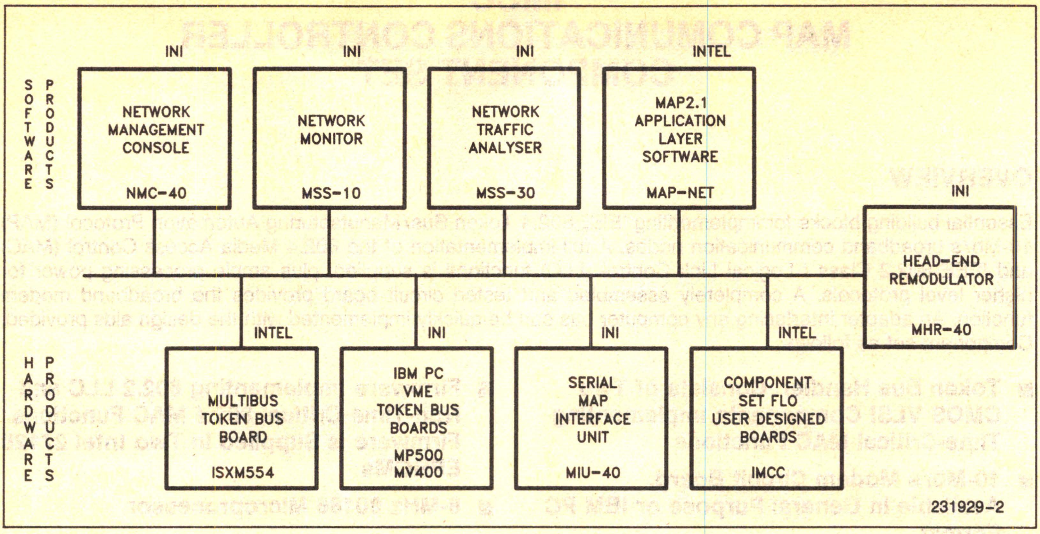


Figure 1. Intel/INI System Environment

HARDWARE ARCHITECTURE

The MCC hardware architecture is based on a dual bus scheme: Control and Storage busses. The main function of the Control bus is to be a local bus for the 80186 microprocessor. The Control bus incorporates the EPROMs with the MAC/LLC firmware and any additional user written code. Through the Control bus the local processor can access the TBH as a slave peripheral. The Storage bus is the interface to the local dynamic RAM. This bus is the output of an arbitration logic that allows multiple ports (local 80186, TBH, System bus) to access the shared DRAM space. The two major merits of this architecture are its performance and modularity.

Performance — is achieved due to the dual bus scheme. The MAC/LLC firmware executes out of the Control bus, which can run concurrently with data transfers from the TBH to the shared DRAM memory space. This architecture is only possible due to the dual-port nature of the TBH controller.

Modularity — As can be seen in Figure 2, the system bus interfacing is done through a shared DRAM window. (System bus slave interface module.) By simply modifying this *slave interface* module, users can adapt the MCC to multiple distinct system busses (backplanes). Also shown is an optional *system bus master* interface. The purpose of this interface is to accommodate added flexibility in multiprocessing busses. This addition allows the local 80186 microprocessor to access the system bus as a master device (this addition is implemented in the Intel iSXM554 board product).

FUNCTIONAL DESCRIPTION

The iMCC implements an intelligent Token Bus/MAP Local Area Network controller. The MCC provides all the components and firmware necessary to build a Token Bus adapter, from the physical to the logical link layers.

The MCC components are:

- **8-MHz 80186:** The main intelligence element of the Token Bus controller. It performs 802.4 MAC non-time-critical functions, 802.2 Class I LLC, and any user desired code.

- **Two 27128 EPROMs:** Carriers of the MCC firmware. They incorporate the non-time-critical functions of the MAC and the 802.2 LLC. About 16 kbytes of the total 32 kb are used for MAC/LLC.
- **Token Bus Handler (TBH):** Implements the time-critical functions of the 802.4 MAC. It incorporates two CMOS integrated circuits which behave as one logical entity. The TBH components have 120 and 144 pins, and use ceramic pin-grid-array packages.
- **10-Mbps Broadband Modem:** Fully compatible with IEEE 802.4 specifications, the modem implements the duobinary AM/PSK encoding scheme, and uses two 6-MHz RF channel pairs for each data direction (Transmit/Receive). Two modem form factors are available, one general purpose and one for the IBM PC Bus. Both modems are powered through their interface cable which is a 34-pin flat ribbon. The general purpose modem can be piggybacked on a Token Bus controller board of the MULTIBUS I form factor.

SYSTEM ENVIRONMENT

MCC is part of a broader Token-Bus/MAP networking system developed jointly by Intel and Industrial Networking Incorporated (INI). It includes building blocks to ensure complete MAP *interconnectability* and *interoperability* at all seven OSI Reference Model layers. Component sets, boards, and software are available from Intel and INI—offering the user a complete solution. Specifically, MCC is the component set that will allow users flexibility in designing their own bus structures and form factors. In addition, the user benefits from the availability of a variety of Intel/INI systems products.

FUNCTIONAL PARTITIONING

The MCC implements a high functionality, high performance MAP/Token Bus controller. An important goal of the MCC architecture is flexibility and upgradeability. The MCC functionality is carefully partitioned between silicon and firmware to achieve the flexibility goals, without sacrificing performance. The time-critical function of the 802.4 protocols are implemented in the TBH silicon for maximum performance. The non-time-critical functions are implemented in firmware which is EPROM based. This approach enables the MCC to track possible 802.4 changes and/or evolutions in an expeditious manner.

As seen in Figure 3, the MCC firmware uses about 16 kbytes of EPROM, and consumes about 10% of the 80186 bandwidth.

THE VIRTUAL TOKEN BUS CONTROLLER (VTBC)

The VTBC is the communications kernel of the MAP/Token Bus controller. The VTBC executes the 802.4 MAC and is composed of the 80186 microprocessor and the TBH. As implied by the name the VTBC is designed to look like a virtual component to

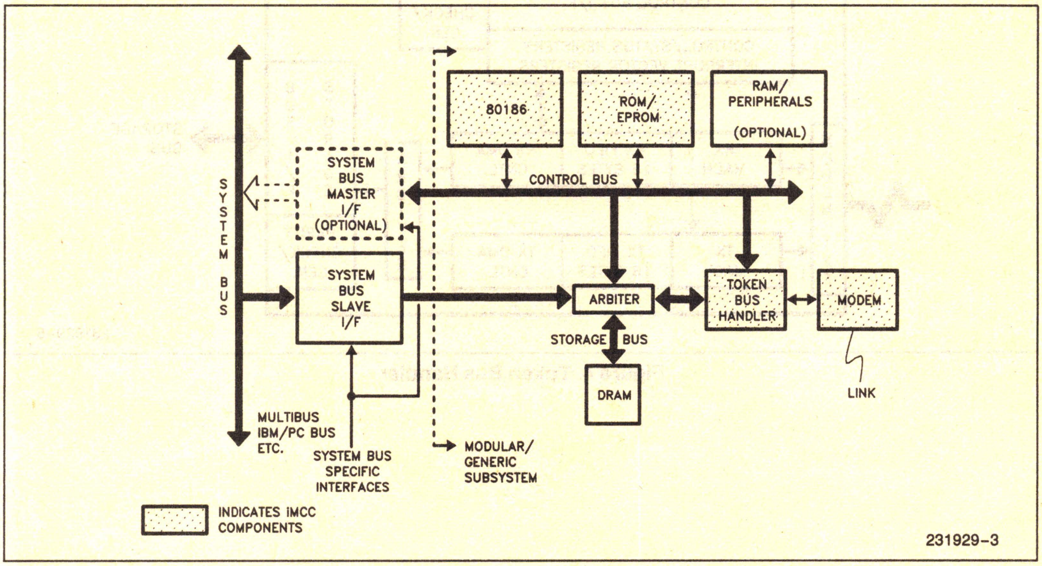


Figure 2. iMCC Hardware Architecture

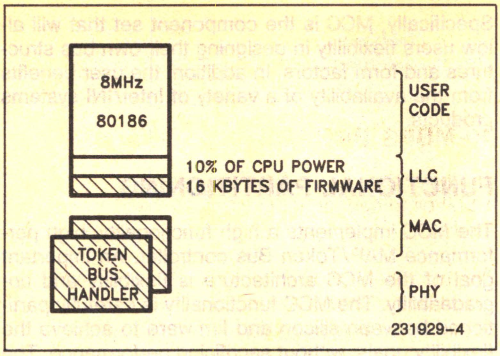


Figure 3. Functional Partitioning

the hardware and software designer. The VTBC provides designers the performance and flexibility necessary to track evolving standards. As described earlier, the time-critical functions are implemented in the TBH; whereas, non-time-critical ones are implemented in firmware for flexibility.

VTBC Functions Provided in Hardware (TBH)

- Framing
- DMA to/from Memory
- Rx Frames Address/Control Filtering
- Automatic Token Passing
- Rotation/Slot Time/Inter Solicit Timers Handling
- Previous/Next Station Address Handling
- Parity Checking for Control/Storage busses

VTBC Function Implemented in Firmware

- MAC State Machine
- Ring Buildup and Maintenance
- All Additional Functions Necessary to Implement Fully the 802.4 MAC Sublayer

FIRMWARE ORGANIZATION

The Data Link firmware executes on the local MCC 80186. It consists of two parts, the MAC Firmware and the LLC Firmware. The MAC Firmware handles all interrupts from the TBH. It provides all functions necessary to implement the IEEE 802.4 Token Passing bus access method. To do this, it must respond to events on the network, select, and execute station responses. The data link provides services to the client's software including the 802.2 Class I sublayer.

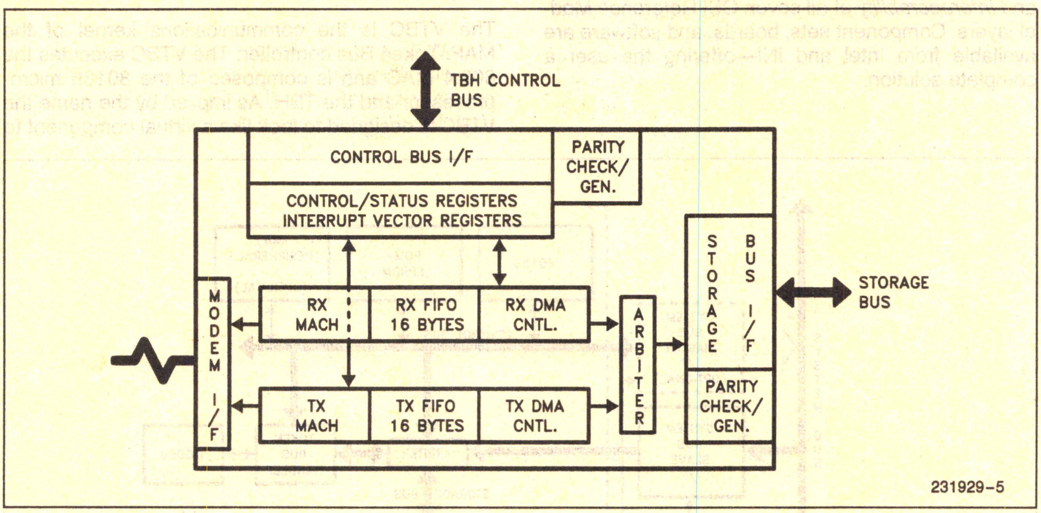


Figure 4. Token Bus Handler

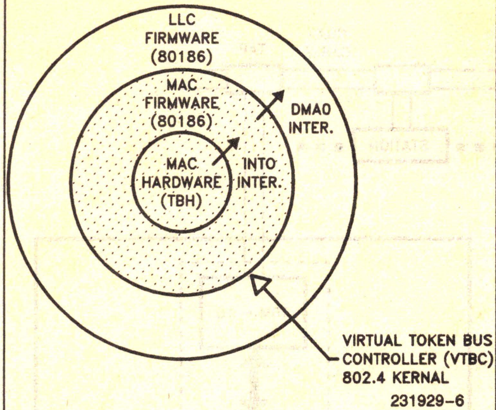


Figure 5. MAC/LLC Partitioning

The MAC firmware is executed only in response to interrupts from the TBH hardware. INT0 is used for this purpose, and it has the highest priority. The MAC firmware complements the TBH in implementing the Access Control Machine (ACM) as specified by 802.4. Together, these elements implement the VTBC which is the MCC 802.4 kernel.

As previously stated, the VTBC acts as a VLSI device, and posts CPU interrupts whenever events need to be reported. This VTBC interrupt is generated in the MAC firmware by an unused interrupt from one of the 80186 internal DMAs (DMA0). Therefore, the VTBC interrupt is *distinct* from the TBH interrupt (INT0) which is generated by the TBH hardware, and effectively internal to the VTBC.

10-MBPS BROADBAND MODEM

The MAP Communications Controller incorporates a 10-Mbps broadband modem to interface with the media.

The broadband modem is implemented on a printed circuit board, and is available in two form factors:

1. IBM PC Bus Form Factor (TBM-12)
2. General Purpose Form Factor (TBM-15)

The modem is designed to operate in association with a Token Bus controller board, and interfaces with that board through a 34-line, flat ribbon cable.

The modem implements the 802.4 duobinary AM/PSK encoding scheme, and uses two 6 MHz RF channels for communication, one pair (12 MHz) in each direction (frequencies used: 6'/FM1'/T/U).

The MCC modem requires a headend remodulator on the network. Transmitted data reaches this device as an amplitude modulated RF signal. The headend remodulates this signal at a different carrier frequency, and sends it to all receivers.

The MCC modem interface is functionally compatible with the MAC-Physical interface specification, originally proposed to the IEEE 802.4 by Motorola (October 1985).

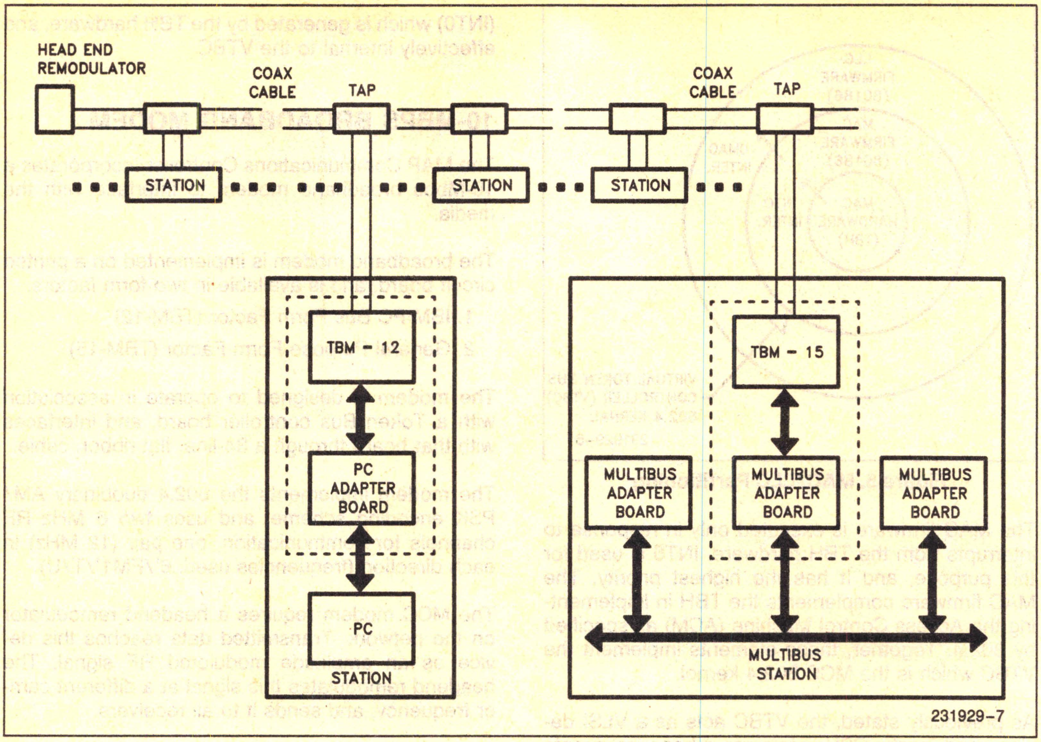


Figure 6. Token Bus Network Example

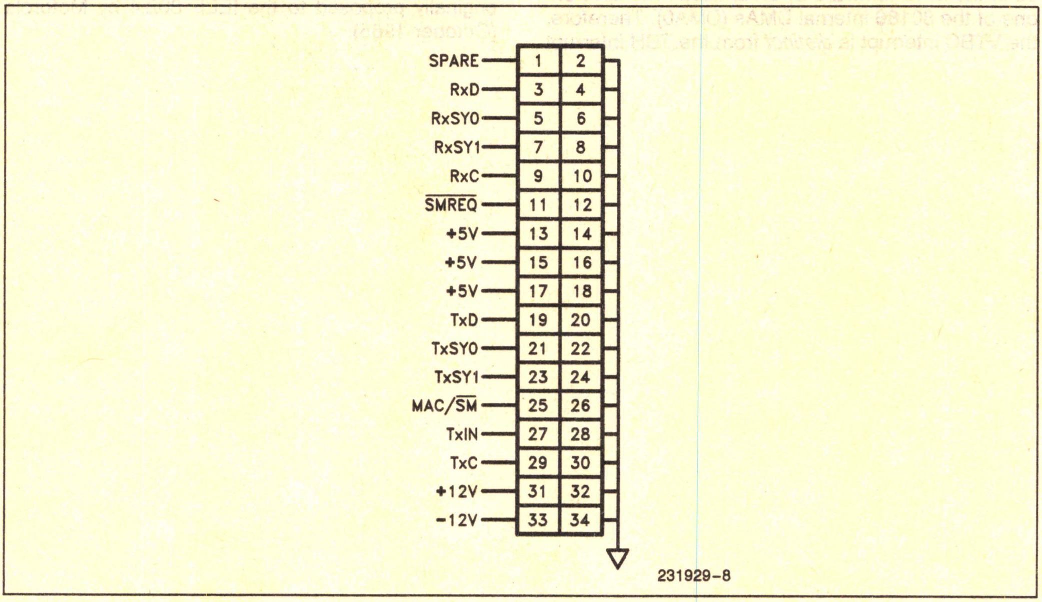


Figure 7. Modem Interface Flat Ribbon Connector

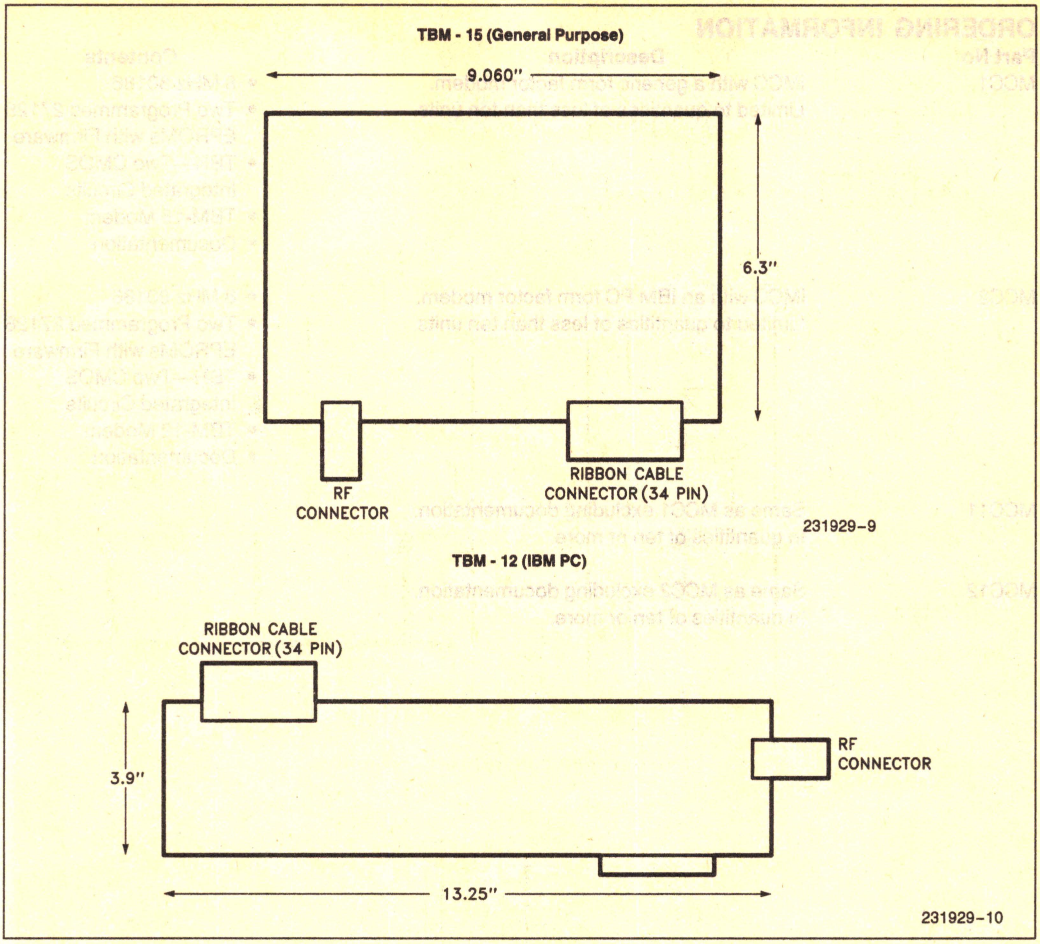


Figure 8. Modem Form Factors

REFERENCE DOCUMENTATION

- iMCC: Firmware Reference Manual
- iMCC: TBM-12, TBM-15 Modem, Data Sheet
- iMCC: AP-301 Token Bus System Design
- iMCC: Token Bus Handler (TBH) Data Sheet

ORDERING INFORMATION

Part No.	Description	Contents
MCC1	iMCC with a generic form factor modem. Limited to quantities of less than ten units.	• 8 MHz 80186 • Two Programmed 27128 EPROMs with Firmware • TBH—Two CMOS Integrated Circuits • TBM-15 Modem • Documentation
MCC2	iMCC with an IBM PC form factor modem. Limited to quantities of less than ten units.	• 8 MHz 80186 • Two Programmed 27128 EPROMs with Firmware • TBH—Two CMOS Integrated Circuits • TBM-12 Modem • Documentation
MCC11	Same as MCC1 excluding documentation. In quantities of ten or more.	
MCC12	Same as MCC2 excluding documentation. In quantities of ten or more.	

Wide Area Networks 2

8251A
PROGRAMMABLE COMMUNICATION INTERFACE

- **Synchronous and Asynchronous Operation**
- **Synchronous 5–8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion**
- **Asynchronous 5–8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling**
- **Synchronous Baud Rate—DC to 64K Baud**
- **Asynchronous Baud Rate—DC to 19.2K Baud**
- **Full-Duplex, Double-Buffered Transmitter and Receiver**
- **Error Detection—Parity, Overrun and Framing**
- **Compatible with an Extended Range of Intel Microprocessors**
- **28-Pin DIP Package**
- **All Inputs and Outputs are TTL Compatible**
- **Available in EXPRESS and Military Versions**

The Intel® 8251A is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel's microprocessor families such as MCS-48, 80, 85, and iAPX-86, 88. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM "bi-sync"). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPTY. The chip is fabricated using Intel's high performance HMOS technology.

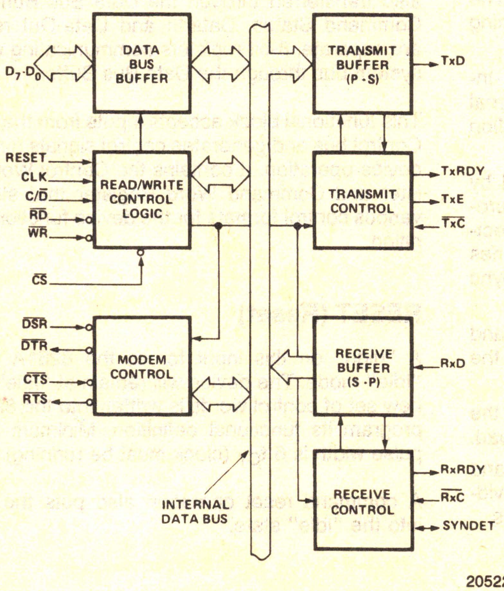


Figure 1. Block Diagram

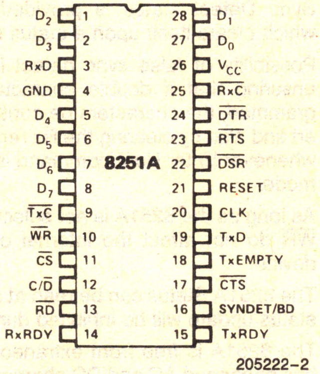


Figure 2. Pin Configuration

November 1986
Order Number: 205222-002

FEATURES AND ENHANCEMENTS

The 8251A is an advanced design of the industry standard USART, the Intel® 8251. The 8251A operates with an extended range of Intel microprocessors and maintains compatibility with the 8251. Familiarization time is minimal because of compatibility and involves only knowing the additional features and enhancements, and reviewing the AC and DC specifications of the 8251A.

The 8251A incorporates all the key features of the 8251 and has the following additional features and enhancements:

- 8251A has double-buffered data paths with separate I/O registers for control, status, Data In, and Data Out, which considerably simplifies control programming and minimizes CPU overhead.
- In asynchronous operations, the Receiver detects and handles "break" automatically, relieving the CPU of this task.
- A refined Rx initialization prevents the Receiver from starting when in "break" state, preventing unwanted interrupts from a disconnected USART.
- At the conclusion of a transmission, TxD line will always return to the marking state unless SBRK is programmed.
- Tx Enable logic enhancement prevents a Tx Disable command from halting transmission until all data previously written has been transmitted. The logic also prevents the transmitter from turning off in the middle of a word.
- When External Sync Detect is programmed, Internal Sync Detect is disabled, and an External Sync Detect status is provided via a flip-flop which clears itself upon a status read.
- Possibility of false sync detect is minimized by ensuring that if double character sync is programmed, the characters be contiguously detected and also by clearing the Rx register to all ones whenever Enter Hunt command is issued in Sync mode.
- As long as the 8251A is not selected, the $\overline{RD}$ and $\overline{WR}$ do not affect the internal operation of the device.
- The 8251A Status can be read at any time but the status update will be inhibited during status read.
- The 8251A is free from extraneous glitches and has enhanced AC and DC characteristics, providing higher speed and better operating margins.
- Synchronous Baud rate from DC to 64K.

FUNCTIONAL DESCRIPTION

General

The 8251A is a Universal Synchronous/Asynchronous Receiver/Transmitter designed for a wide range of Intel microcomputers such as 8048, 8080, 8085, 8086 and 8088. Like other I/O devices in a microcomputer system, its functional configuration is programmed by the system's software for maximum flexibility. The 8251A can support most serial data techniques in use, including IBM "bi-sync".

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state bidirectional, 8-bit buffer is used to interface the 8251A to the system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer. The Command Status, Data-In and Data-Out registers are separate, 8-bit registers communicating with the system bus through the Data Bus Buffer.

This functional block accepts inputs from the system Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for the device functional definition.

RESET (Reset)

A "high" on this input forces the 8251A into an "Idle" mode. The device will remain at "Idle" until a new set of control words is written into the 8251A to program its functional definition. Minimum RESET pulse width is 6 t_{CY} (clock must be running).

A command reset operation also puts the device into the "Idle" state.

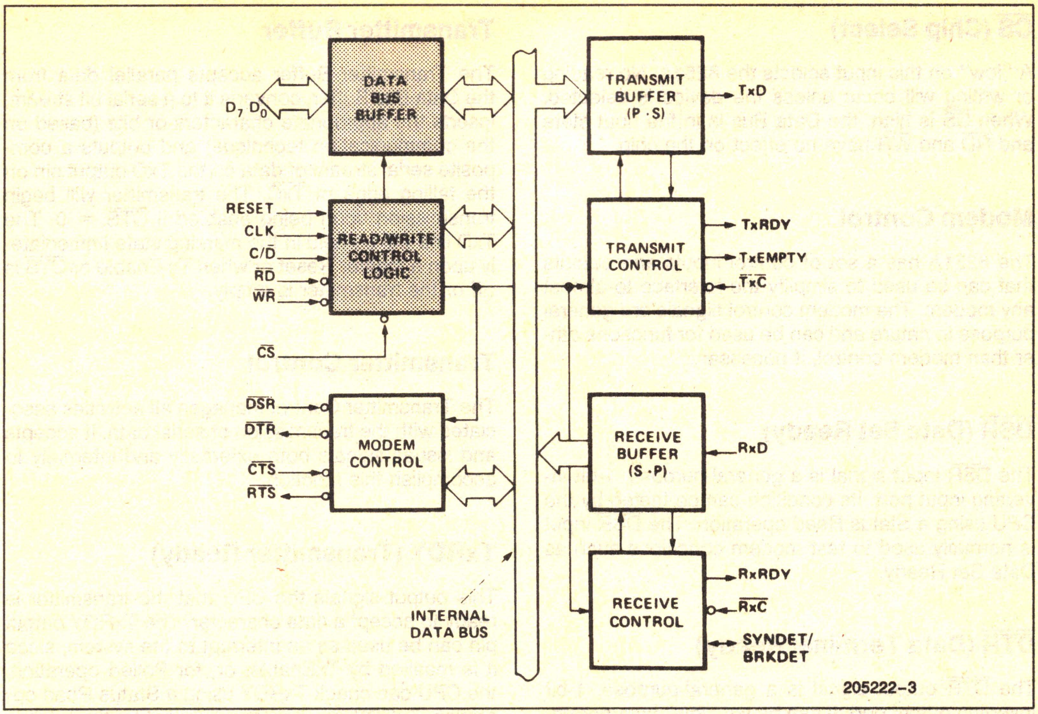


Figure 3. 8251A Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

$\overline{WR}$ (Write)

A "low" on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

$\overline{RD}$ (Read)

A "low" on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251A DATA → DATA BUS
0	1	0	0	DATA BUS → 8251A DATA
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	1	1	0	DATA BUS → 3-STATE
X	X	X	1	DATA BUS → 3-STATE

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs, informs the 8251A that the word on the Data Bus is either a data character, control word or status information.

1 = CONTROL/STATUS; 0 = DATA.

$\overline{\text{CS}}$ (Chip Select)

A "low" on this input selects the 8251A. No reading or writing will occur unless the device is selected. When $\overline{\text{CS}}$ is high, the Data Bus is in the float state and $\overline{\text{RD}}$ and $\overline{\text{WR}}$ have no effect on the chip.

Modem Control

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. The modem control signals are general purpose in nature and can be used for functions other than modem control, if necessary.

$\overline{\text{DSR}}$ (Data Set Ready)

The $\overline{\text{DSR}}$ input signal is a general-purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The $\overline{\text{DSR}}$ input is normally used to test modem conditions such as Data Set Ready.

$\overline{\text{DTR}}$ (Data Terminal Ready)

The $\overline{\text{DTR}}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{\text{DTR}}$ output signal is normally used for modem control such as Data Terminal Ready.

$\overline{\text{RTS}}$ (Request to Send)

The $\overline{\text{RTS}}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{\text{RTS}}$ output signal is normally used for modem control such as Request to Send.

$\overline{\text{CTS}}$ (Clear to Send)

A "low" on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a "one". If either a Tx Enable off or $\overline{\text{CTS}}$ off condition occurs while the Tx is in operation, the Tx will transmit all the data in the USART, written prior to Tx Disable command before shutting down.

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of $\overline{\text{TxC}}$. The transmitter will begin transmission upon being enabled if $\overline{\text{CTS}} = 0$. The TxD line will be held in the marking state immediately upon a master Reset or when Tx Enable or $\overline{\text{CTS}}$ is off or the transmitter is empty.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. The TxRDY output pin can be used as an interrupt to the system, since it is masked by TxEnable; or, for Polled operation, the CPU can check TxRDY using a Status Read operation. TxRDY is automatically reset by the leading edge of $\overline{\text{WR}}$ when a data character is loaded from the CPU.

Note that when using the Polled operation, the TxRDY status bit is *not* masked by TxEnable, but will only indicate the Empty/Full Status of the Tx Data Input Register.

TxE (Transmitter Empty)

When the 8251A has no characters to send, the TxEMPTY output will go "high". It resets upon receiving a character from CPU if the transmitter is enabled. TxEMPTY remains high when the transmitter is disabled. TxEMPTY can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplex operational mode.

In the Synchronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be or are being transmitted automatically as "fillers". Tx EMPTY does not go low when the SYNC characters are being shifted out.

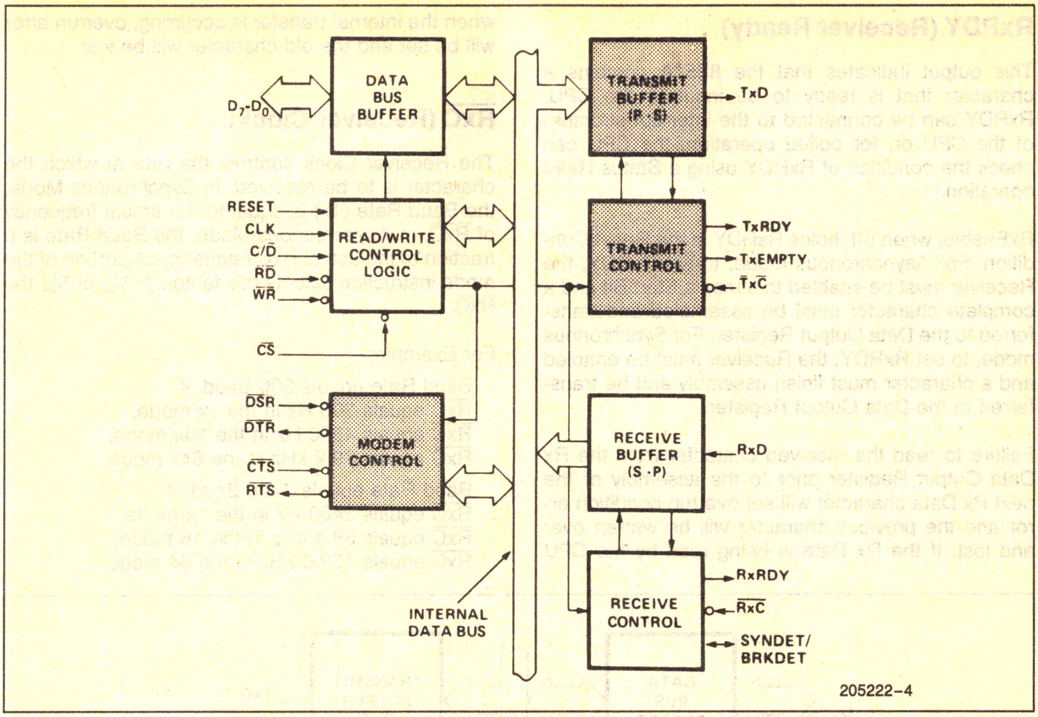


Figure 4. 8251A Block Diagram Showing Modem and Transmitter Buffer and Control Functions

T̄x̄C̄ (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the T̄x̄C̄ frequency. In Asynchronous transmission mode, the baud rate is a fraction of the actual T̄x̄C̄ frequency. A portion of the mode instruction selects this factor; it can be 1, 1/16 or 1/64 the T̄x̄C̄.

For Example:

If Baud Rate equals 110 Baud,
T̄x̄C̄ equals 110 Hz in the 1x mode.
T̄x̄C̄ equals 1.72 kHz in the 16x mode.
T̄x̄C̄ equals 7.04 kHz in the 64x mode.

The falling edge of T̄x̄C̄ shifts the serial data out of the 8251A.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to RxD pin, and is clocked in on the rising edge of R̄x̄C̄.

Receiver Control

This functional block manages all receiver-related activities which consists of the following features.

The RxD initialization circuit prevents the 8251A from mistaking an unused input line for an active low data line in the "break condition". Before starting to receive serial characters on the RxD line, a valid "1" must first be detected after a chip master Reset. Once this has been determined, a search for a valid low (Start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master Reset.

The False Start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the normal center of the Start bit (RxD = low).

Parity error detection sets the corresponding status bit.

The Framing Error status bit is set if the Stop bit is absent at the end of the data byte (asynchronous mode).

RxRDY (Receiver Ready)

This output indicates that the 8251A contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or, for polled operation, the CPU can check the condition of RxRDY using a Status Read operation.

RxEnable, when off, holds RxRDY in the Reset Condition. For Asynchronous mode, to set RxRDY, the Receiver must be enabled to sense a Start Bit and a complete character must be assembled and transferred to the Data Output Register. For Synchronous mode, to set RxRDY, the Receiver must be enabled and a character must finish assembly and be transferred to the Data Output Register.

Failure to read the received character from the Rx Data Output Register prior to the assembly of the next Rx Data character will set overrun condition error and the previous character will be written over and lost. If the Rx Data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the Baud Rate (1x) is equal to the actual frequency of $\overline{RxC}$. In Asynchronous Mode, the Baud Rate is a fraction of the actual $\overline{RxC}$ frequency. A portion of the mode instruction selects this factor: 1, $1/16$ or $1/64$ the $\overline{RxC}$.

For Example:

Baud Rate equals 300 Baud, if
$\overline{RxC}$ equals 300 Hz in the 1x mode;
$\overline{RxC}$ equals 4800 Hz in the 16x mode;
$\overline{RxC}$ equals 19.2 kHz in the 64x mode.

Baud Rate equals 2400 Baud, if
$\overline{RxC}$ equals 2400 Hz in the 1x mode;
$\overline{RxC}$ equals 38.4 kHz in the 16 mode;
$\overline{RxC}$ equals 153.6 kHz in the 64 mode.

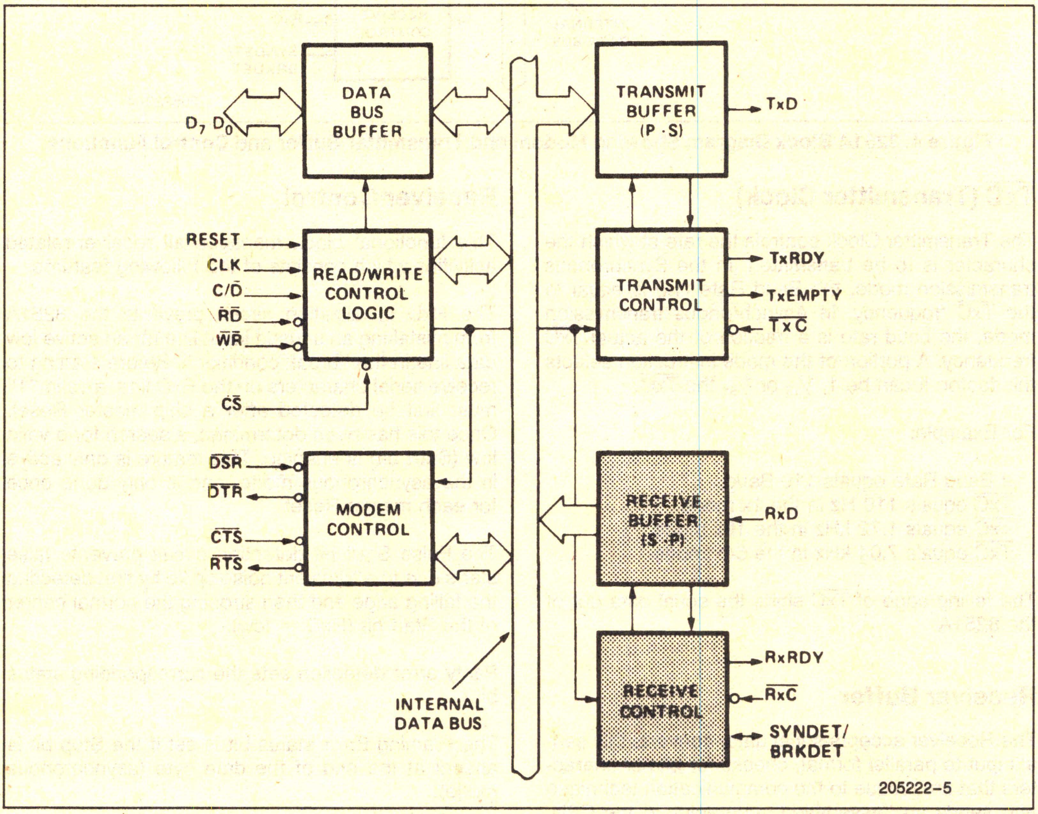


Figure 5. 8251A Block Diagram Showing Receiver Buffer and Control Functions

Data is sampled into the 8251A on the rising edge of $\overline{RxC}$.

NOTE:
In most communication systems, the 8251A will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

SYNDET (SYNC Detect/ BRKDET Break Detect)

This pin is used in Synchronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, Internal SYNC Detect is disabled.

BREAK (Async Mode Only)

This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits). Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset or Rx Data returning to a "one" state.

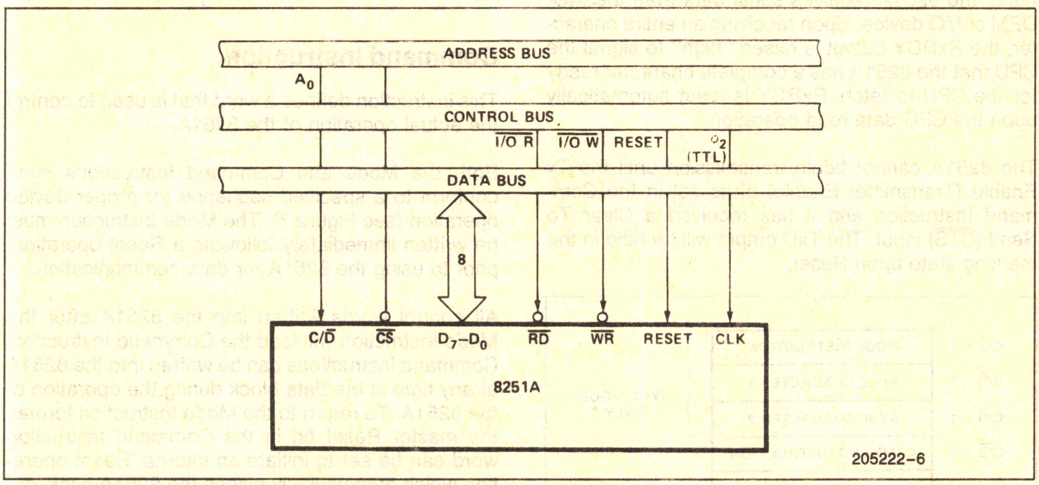


Figure 6. 8251A Interface to 8080 Standard System Bus

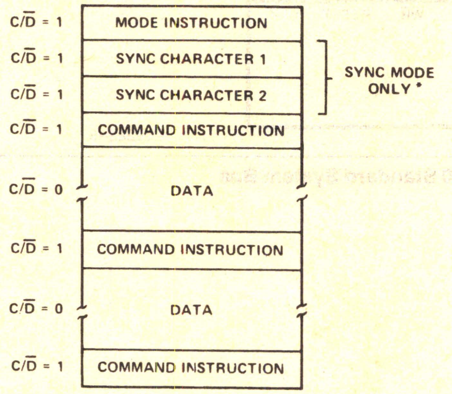
DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251A is programmed by the system's software. A set of control words must be sent out by the CPU to initialize the 8251A to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD/OFF PARITY, etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251A is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251A is ready to receive a data character from the CPU. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251A. On the other hand, the 8251A receives serial data from the MODEM or I/O device. Upon receiving an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251A has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU data read operation.

The 8251A cannot begin transmission until the Tx Enable (Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send ($\overline{CTS}$) input. The TxD output will be held in the marking state upon Reset.



205222-7

*The second sync character is skipped if mode instruction has programmed the 8251A to single character sync mode. Both sync characters are skipped if mode instruction has programmed the 8251A to async mode.

Figure 7. Typical Data Block

Programming the 8251A

Prior to starting data transmission or reception, the 8251A must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251A and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

This instruction defines the general operational characteristics of the 8251A. It must follow a Reset operation (internal or external). Once the Mode Instruction has been written into the 8251A by the CPU, SYNC characters or Command Instructions may be written.

Command Instruction

This instruction defines a word that is used to control the actual operation of the 8251A.

Both the Mode and Command Instructions must conform to a specified sequence for proper device operation (see Figure 7). The Mode Instruction must be written immediately following a Reset operation, prior to using the 8251A for data communication.

All control words written into the 8251A after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251A at any time in the data block during the operation of the 8251A. To return to the Mode Instruction format, the master Reset bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251A back into the Mode Instruction format. Command Instructions must follow the Mode Instruction or Sync characters.

Mode Instruction Definition

The 8251A can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251A, the designer can best view the device as two separate components, one Asynchronous and the other Synchronous, sharing the same package. The format definition can be changed only after a master chip Reset. For explanation purposes the two formats will be isolated.

NOTE:
When parity is enabled it is not considered as one of the data bits for the purpose of programming word length. The actual parity bit received on the Rx Data line cannot be read on the Data Bus. In the case of a programmed character length of less than 8 bits, the least significant Data Bus bits will hold the data; unused bits are "don't care" when writing data to the 8251A, and will be "zeros" when reading the data from the 8251A.

Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251A automatically adds a Start bit (low level) followed by the data bits (least significant bit first), and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, $1/16$, or $1/64$ that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

When no data characters have been loaded into the 8251A the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center (16X or 64X mode only). If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter thus locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of the $\overline{RxC}$. If a low level is detected as the STOP bit the Framing Error flag will be set. The STOP bit signals the end of a character. Note that the *receiver* requires only *one* stop bit, regardless of the number of stop bits programmed. This character is then loaded into the parallel I/O buffer of the 8251A. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN Error flag

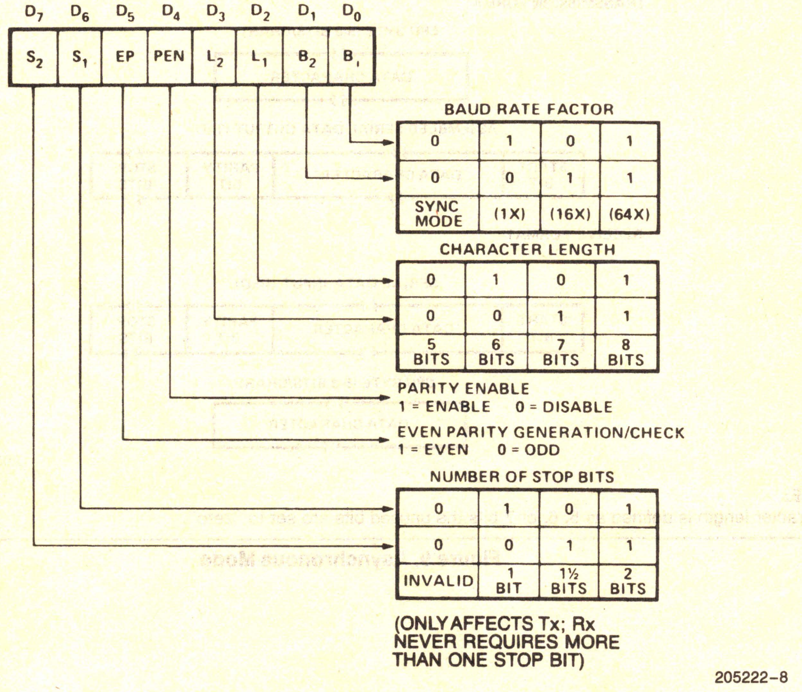


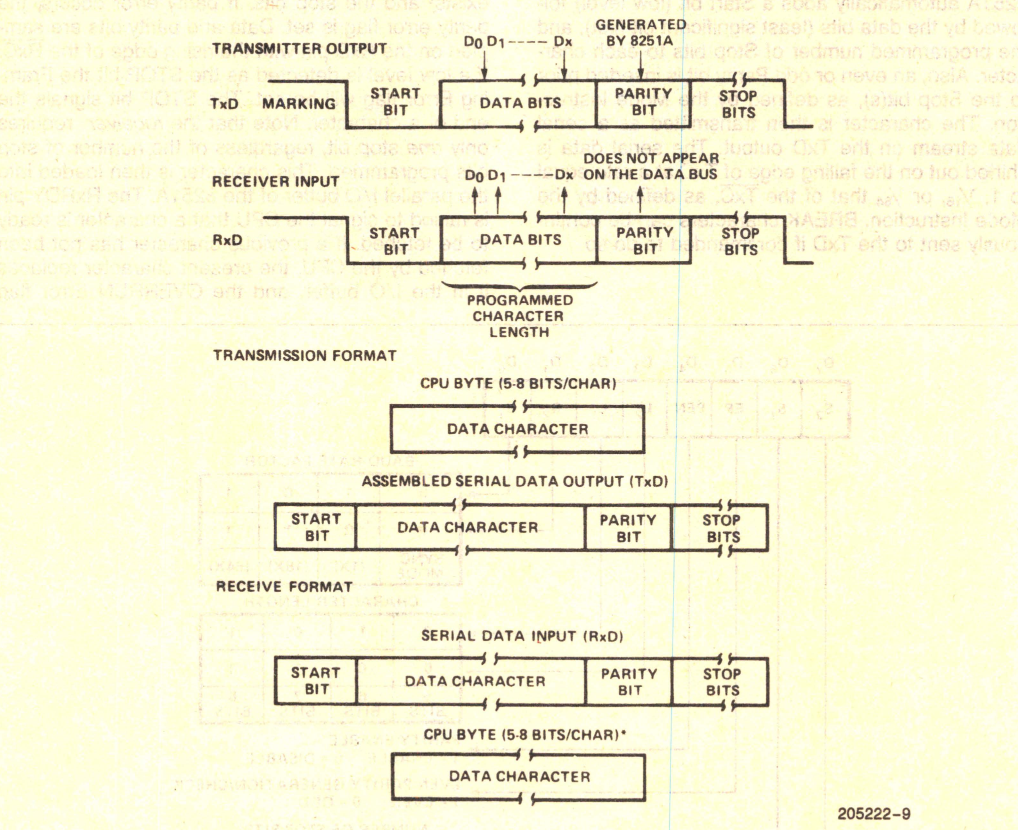
Figure 8. Mode Instruction Format, Asynchronous Mode

is raised (thus the previous character is lost). All of the error flags can be reset by an Error Reset Instruction. The occurrence of any of these errors will not affect the operation of the 8251A.

Synchronous Mode (Transmission)

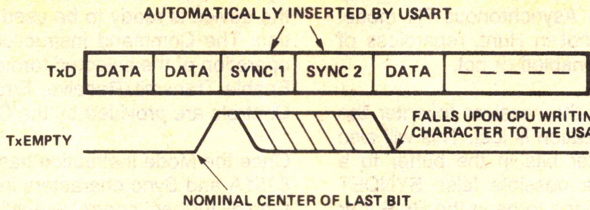
The TxD output is continuously high until the CPU sends its first character to the 8251A which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at the TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251A with a data character before the 8251A Transmitter Buffers become empty, the SYNC characters (or character if in single SYNC character mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251A is empty and SYNC characters are being sent out. TxEMPTY does not go low when the SYNC is being shifted out (see figure below). The TxEMPTY pin is internally reset by a data character being written into the 8251A.



***NOTE:**
If character length is defined as 5, 6, or 7 bits the unused bits are set to "zero".

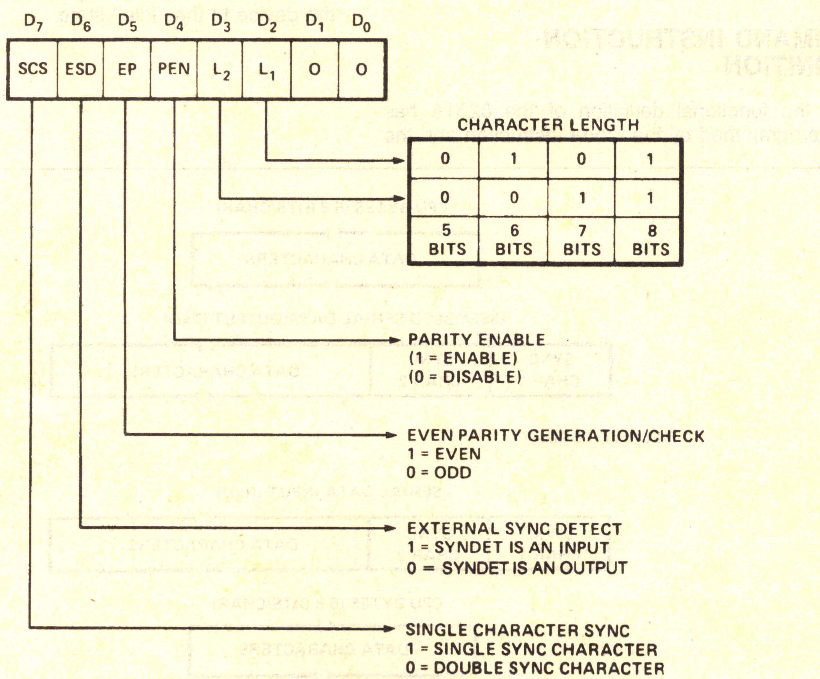
Figure 9. Asynchronous Mode



Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the SYNC mode has been programmed, ENTER HUNT command should be included in the first command instruction word written. Data on the RxD pin is then sampled on the rising edge of $\overline{RxC}$. The content of the Rx buffer is compared at every bit boundary with the first SYNC character until a match occurs. If the 8251A has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ. If parity is programmed, SYNDET will not be set until the middle of the parity bit instead of the middle of the last data bit.

In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin, thus forcing the 8251A out of the HUNT mode. The high level can be removed after one $\overline{RxC}$ cycle. An ENTER HUNT command has no effect in the asynchronous mode of operation.



NOTE:
In external sync mode, programming double character sync will affect only the Tx.

205222-11

Figure 10. Mode Instruction Format, Synchronous Mode

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode. Parity is checked when not in Hunt, regardless of whether the Receiver is enabled or not.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost. This will also set all the used character bits in the buffer to a "one," thus preventing a possible false SYNDET caused by data that happens to be in the Rx Buffer at ENTER HUNT time. Note that the SYNDET F/F is reset at each Status Read, regardless of whether internal or external SYNC has been programmed. This does not cause the 8251A to return to the HUNT mode. When in SYNC mode, but not in HUNT, Sync Detection is still functional, but only occurs at the "known" word boundaries. Thus, if one Status Read indicates SYNDET and a second Status Read also indicates SYNDET, then the programmed SYNDET characters have been received since the previous Status Read. (If double character sync has been programmed, then both sync characters have been contiguously received to gate a SYNDET indication). When external SYNDET mode is selected, internal Sync Detect is disabled, and the SYNDET F/F may be set at any bit boundary.

COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251A has been programmed by the Mode Instruction and the Sync characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command instruction.

Once the Mode Instruction has been written into the 8251A and Sync characters inserted, of necessary, then all further "control writes" ($C/\overline{D}$ = 1) will load a Command Instruction. A Reset Operation (internal or external) will return the 8251A to the Mode Instruction format.

NOTE:
Internal Reset on Power-up:

When power is first applied, the 8251A may come up in the Mode, Sync character or Command format. To guarantee that the device is in the Command Instruction format before the Reset command is issued, it is safest to execute the worst-case initialization sequence (sync mode with two sync characters). Loading three 00Hs consecutively into the device with $C/\overline{D}$ = 1 configures sync operation and writes two dummy 00H sync characters. An Internal Reset command (40H) may then be issued to return the device to the "idle" state.

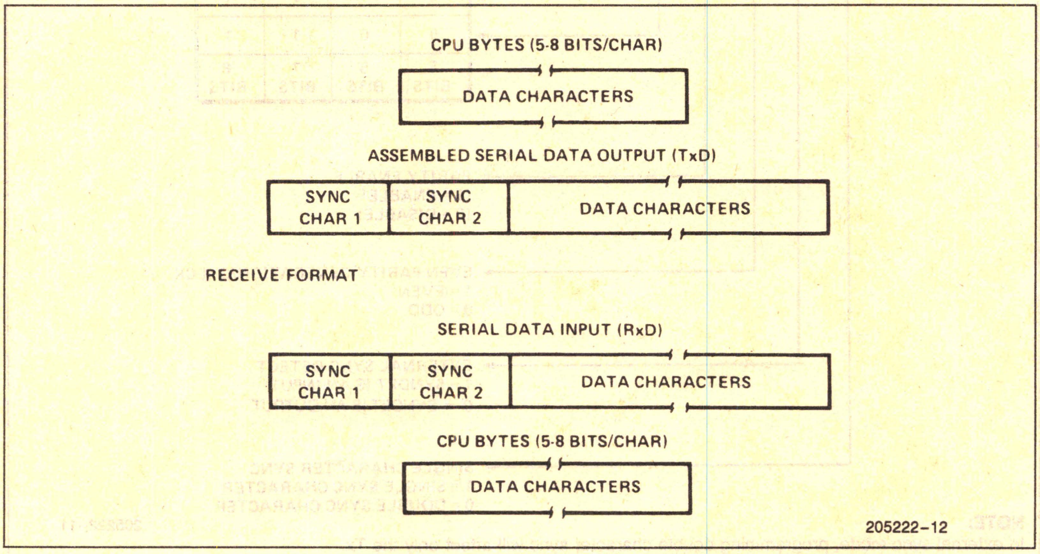
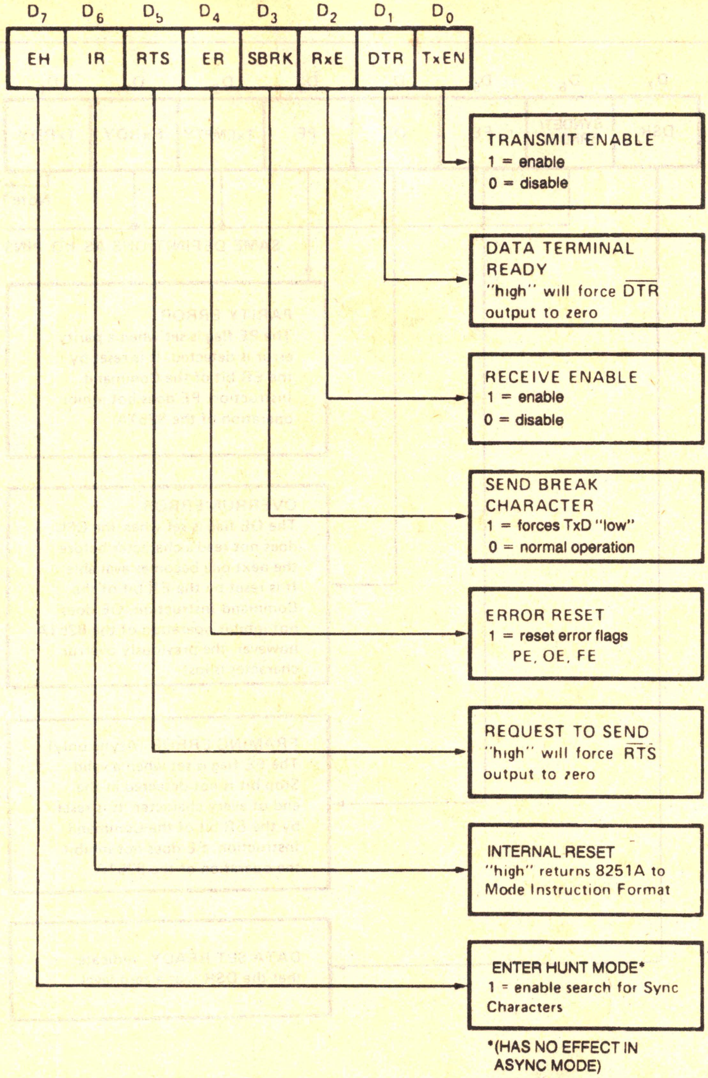


Figure 11. Data Format, Synchronous Mode



NOTE:
Error Reset must be performed whenever RxEnable and Enter Hunt are programmed.

Figure 12. Command Instruction Format

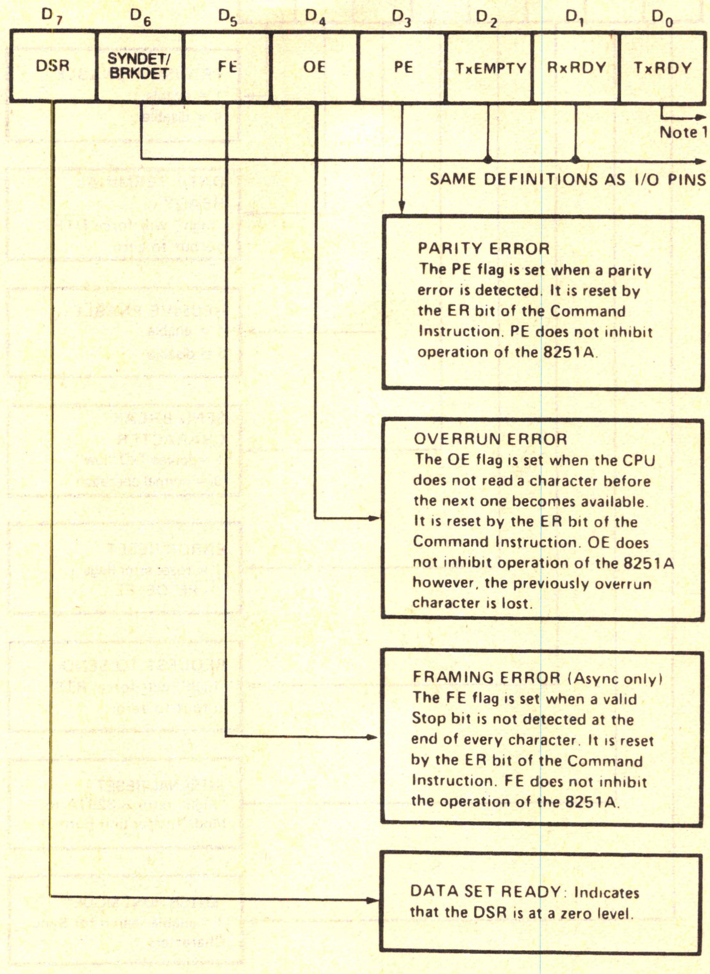
STATUS READ DEFINITION

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251A has facilities that allow the programmer to "read" the status of the device at any time during the functional operation. (Status update is inhibited during status read.)

A normal "read" command is issued by the CPU with C/$\overline{D}$ = 1 to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251A can be used in a completely polled or interrupt-driven environment. TxRDY is an exception.

Note that status update can have a maximum delay of 28 clock periods from the actual event affecting the status.



NOTE:
1. TxRDY status bit has different meanings from the TxRDY output pin. The former is not conditioned by $\overline{CTS}$ and TxEN; the latter is conditioned by both $\overline{CTS}$ and TxEN.
i.e. TxRDY status bit = DB Buffer Empty
TxRDY pin out = DB Buffer Empty • ($\overline{CTS}$ = 0) • (TxEN = 1)

Figure 13. Status Read Format

APPLICATIONS OF THE 8251A

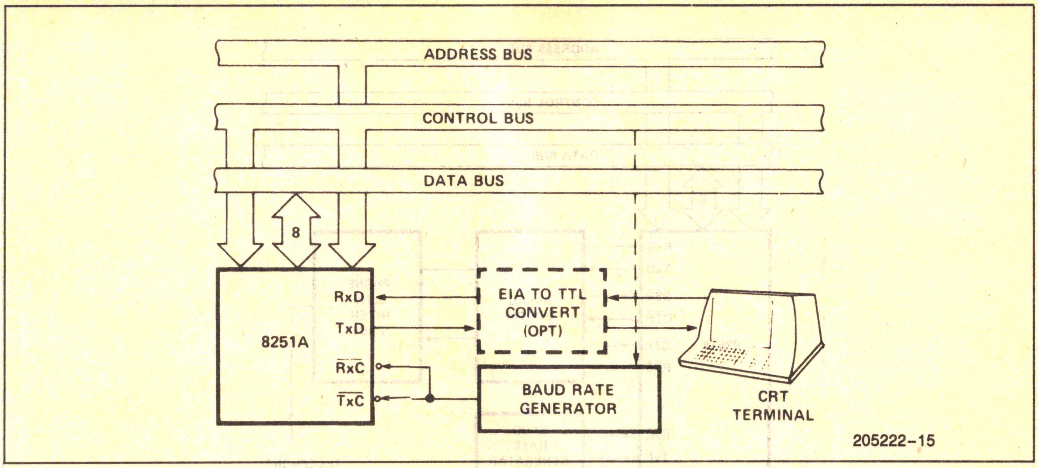


Figure 14. Asynchronous Serial Interface to CRT Terminal, DC—9600 Baud

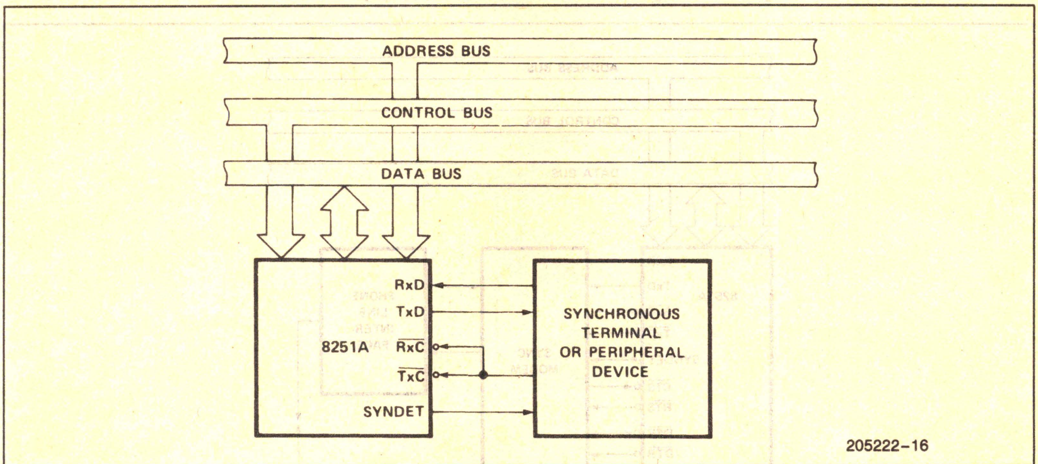


Figure 15. Synchronous Interface to Terminal or Peripheral Device

APPLICATIONS OF THE 8251A (Continued)

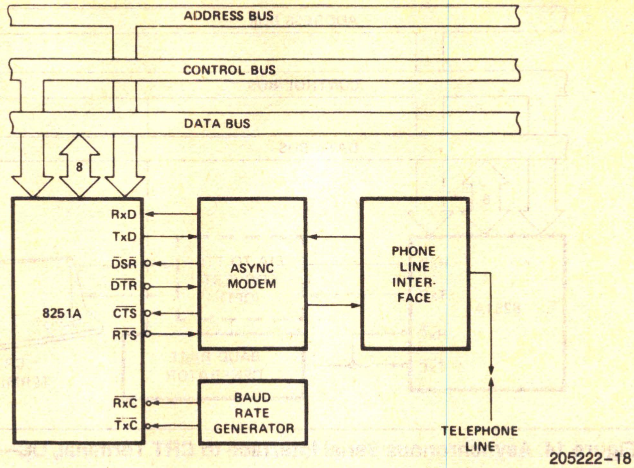


Figure 16. Asynchronous Interface to Telephone Lines

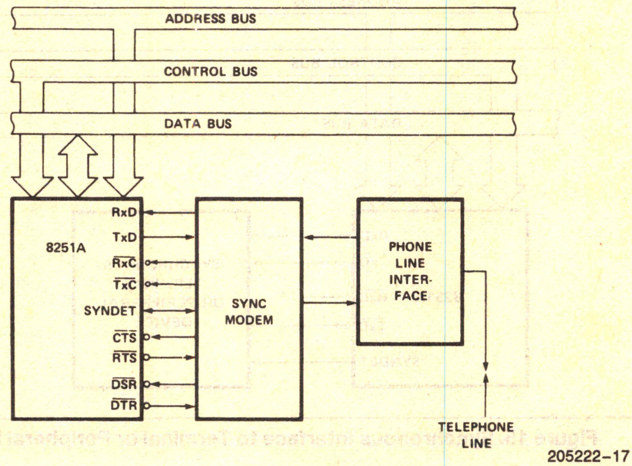


Figure 17. Synchronous Interface to Telephone Lines

NOTES:

1. AC timings measured $V_{OH} = 2.0$ $V_{OL} = 0.8$, and with load circuit of Figure 18.
2. Chip Select (CS) and Command/Data (C/D) are considered as Addresses.
3. Assumes that Address is valid before $R_D \downarrow$.
4. This recovery time is for Mode Initialization only. Write Data is allowed only when TxRDY = 1. Recovery Time between Writes for Asynchronous Mode is 8 t_{CY} and for Synchronous Mode is 16 t_{CY}.
5. The TxC and RxC frequencies have the following limitations with respect to CLK: For 1x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(30\ t_{CY})$: For 16x and 64x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(4.5\ t_{CY})$. This applies to Baud Rates less than or equal to 64K Baud.
6. Reset Pulse Width = 6 t_{CY} minimum; System clock must be running during Reset.
7. Status update can have a maximum delay of 28 clock periods from the event affecting the status.
8. In external sync mode the tes spec. requires the ratio of the system clock (clock) to receive or transmit bit ratios to be greater than 34.
9. A float is defined as the point where the data bus falls below a logic 1 (2.0V @ I_{OH} limit) or rises above a Logic 0 (0.8V @ I_{OL} limit).

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ......0°C to 70°C
Storage Temperature ....−65°C to +150°C
Voltage on Any Pin
with Respect to Ground..........−0.5V to +7V
Power Dissipation 1W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5.0V \pm 10\%$, GND = 0V*

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -400\ \mu A$
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{IL}	Input Leakage		±10	μA	$V_{IN} = V_{CC}$ to 0.45V
I_{CC}	Power Supply Current		100	ma	All Outputs = High

CAPACITANCE $T_A = 25°C$, V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Unit	Test Conditions
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5.0V \pm 10\%$, GND = 0V*

Bus Parameters (Note 1)

READ CYCLE

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AR}	Address Stable Before $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	0		ns	(Note 2)
t_{RA}	Address Hold Time for $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	0		ns	(Note 2)
t_{RR}	$\overline{READ}$ Pulse Width	250		ns	
t_{RD}	Data Delay from $\overline{READ}$		200	ns	3, C_L = 150 pF
t_{DF}	$\overline{READ}$ to Data Floating	10	100	ns	(Note 1, 9)

WRITE CYCLE

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	Address Stable Before $\overline{WRITE}$	0		ns	
t_{WA}	Address Hold Time for $\overline{WRITE}$	0		ns	
t_{WW}	$\overline{WRITE}$ Pulse Width	250		ns	
t_{DW}	Data Set-Up Time for $\overline{WRITE}$	150		ns	
t_{WD}	Data Hold Time for $\overline{WRITE}$	20		ns	
t_{RV}	Recovery Time Between WRITES	6		t_{CY}	(Note 4)

A.C. CHARACTERISTICS (Continued)

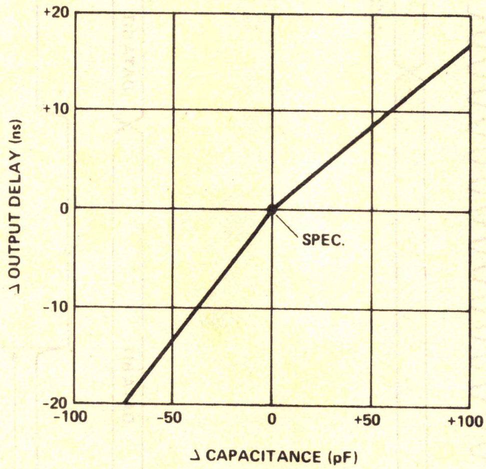
OTHER TIMINGS

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CY}	Clock Period	320	1350	ns	(Note 5, 6)
t_{ϕ}	Clock High Pulse Width	120	$t_{CY}-90$	ns	
$\overline{t_{\phi}}$	Clock Low Pulse Width	90		ns	
t_R, t_F	Clock Rise and Fall Time		20	ns	
t_{DTx}	TxD Delay from Falling Edge of $\overline{TxC}$		1	µs	
f_{Tx}	Transmitter Input Clock Frequency 1x Baud Rate 16x Baud Rate 64x Baud Rate	 DC DC DC	 64 310 615	 kHz kHz kHz	
t_{TPW}	Transmitter Input Clock Pulse Width 1x Baud Rate 16x and 64x Baud Rate	 12 1		 t_{CY} t_{CY}	
t_{TPD}	Transmitter Input Clock Pulse Delay 1x Baud Rate 16x and 64x Baud Rate	 15 3		 t_{CY} t_{CY}	
f_{Rx}	Receiver Input Clock Frequency 1x Baud Rate 16x Baud Rate 64x Baud Rate	 DC DC DC	 64 310 615	 kHz kHz kHz	
t_{RPW}	Receiver Input Clock Pulse Width 1x Baud Rate 16x and 64x Baud Rate	 12 1		 t_{CY} t_{CY}	
t_{RPD}	Receiver Input Clock Pulse Delay 1x Baud Rate 16x and 64x Baud Rate	 15 3		 t_{CY} t_{CY}	
t_{TxRDY}	TxRDY Pin Delay from Center of Last Bit		14	t_{CY}	(Note 7)
$t_{TxRDY\ CLEAR}$	TxRDY ↓ from Leading Edge of $\overline{WR}$		400	ns	(Note 7)
t_{RxRDY}	RxRDY Pin Delay from Center of Last Bit		26	t_{CY}	(Note 7)
$t_{RxRDY\ CLEAR}$	RxRDY ↓ from Leading Edge of $\overline{RD}$		400	ns	(Note 7)
t_{IS}	Internal SYNDET Delay from Rising Edge of $\overline{RxC}$		26	t_{CY}	(Note 7)
t_{ES}	External SYNDET Set-Up Time After Rising Edge of $\overline{RxC}$	16 t_{CY}	$t_{RPD}-t_{CY}$	ns	(Note 7)
$t_{TxEMPTY}$	TxEMPTY Delay from Center of Last Bit		20	t_{CY}	(Note 7)
t_{WC}	Control Delay from Rising Edge of WRITE (TxEn, $\overline{DTR}$, $\overline{RTS}$)		8	t_{CY}	(Note 7)
t_{CR}	Control to READ Set-Up Time ($\overline{DSR}$, $\overline{CTS}$)	20		t_{CY}	(Note 7)

***NOTE:**
For Extended Temperature EXPRESS, use MIL 8251A electrical parameters.

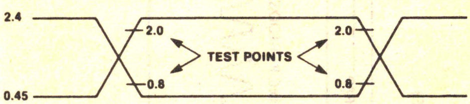
A.C. CHARACTERISTICS (Continued)

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (pF)



205222-19

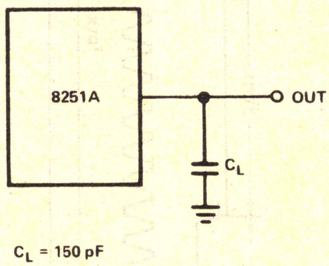
A.C. TESTING INPUT, OUTPUT WAVEFORM



205222-20

AC Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

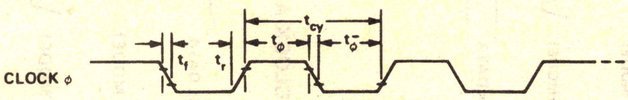


205222-21

Figure 18

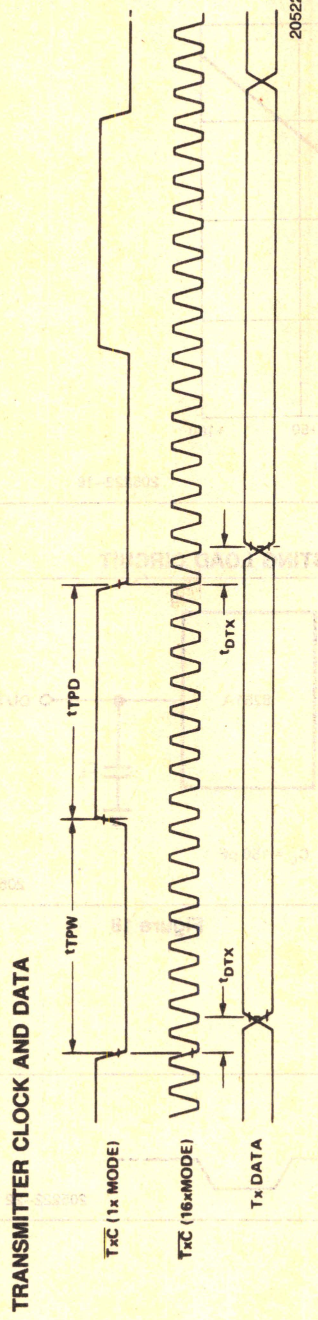
WAVEFORMS

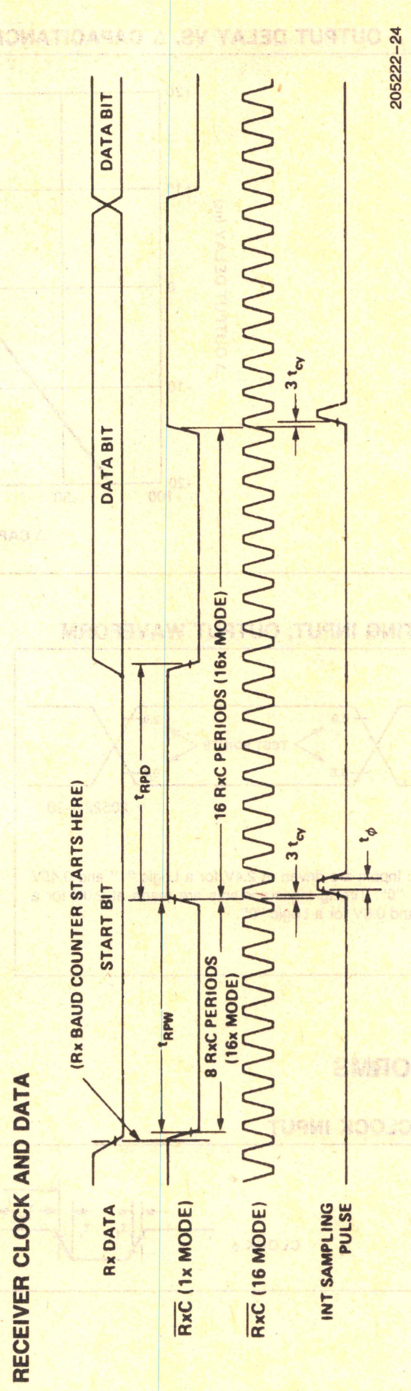
SYSTEM CLOCK INPUT



205222-22

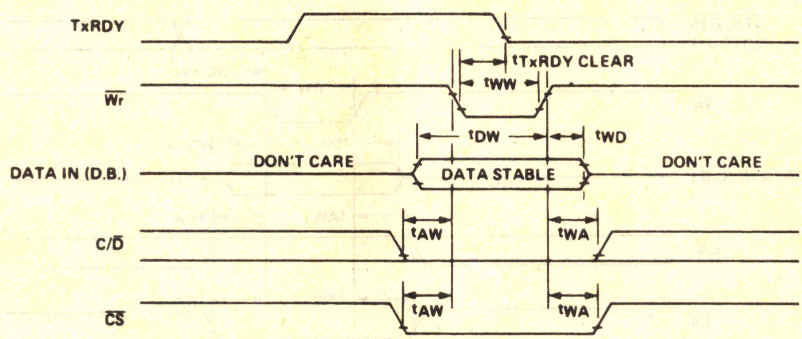
WAVEFORMS (Continued)





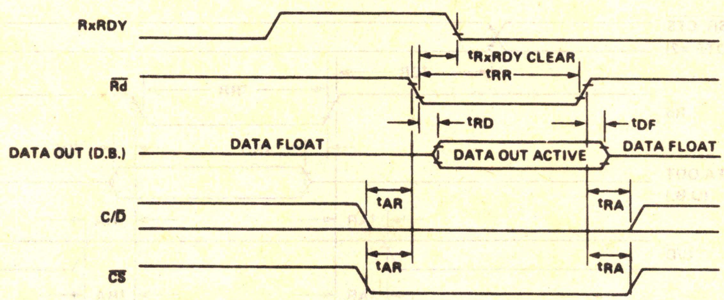
WAVEFORMS (Continued)

WRITE DATA CYCLE (CPU → USART)



205222-25

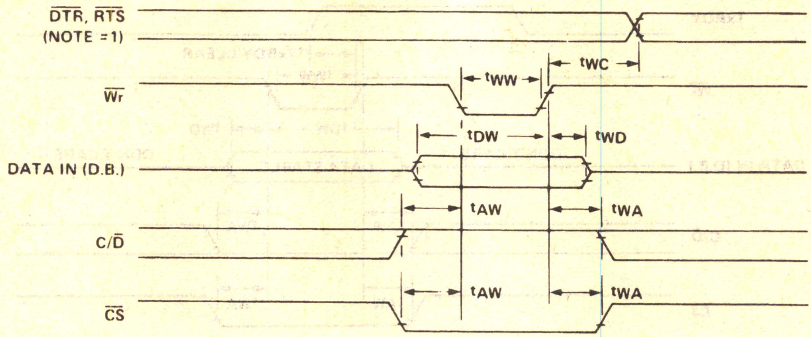
READ DATA CYCLE (CPU ← USART)



205222-26

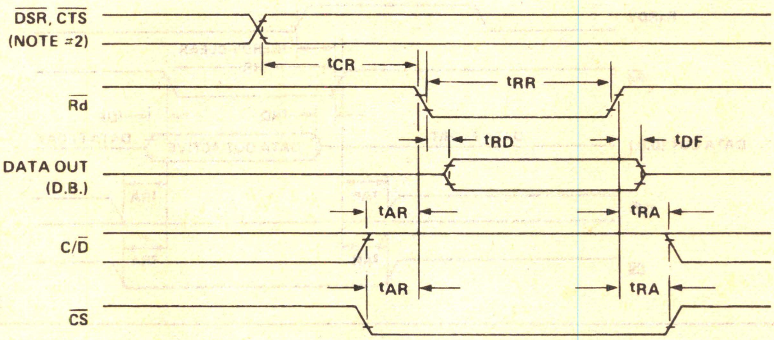
WAVEFORMS (Continued)

WRITE CONTROL OR OUTPUT PORT CYCLE (CPU → USART)



205222-27

READ CONTROL OR INPUT PORT (CPU ← USART)

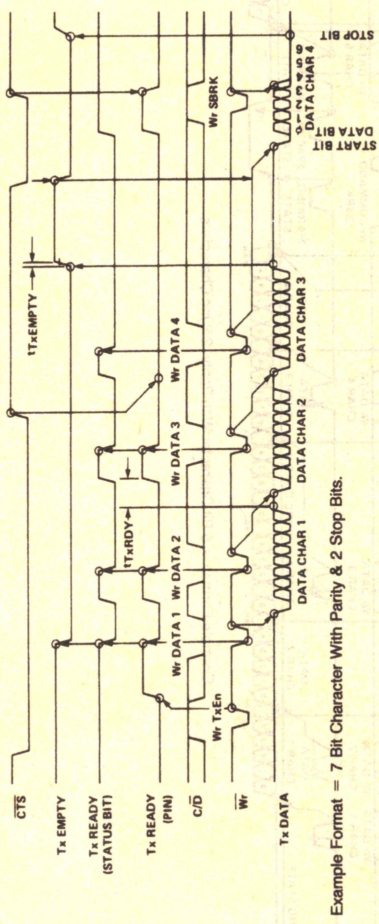


205222-28

NOTES:
1. T_{WC} includes the response timing of a control byte.
2. T_{CR} includes the effect of CTS on the TxENBL circuitry.

WAVEFORMS (Continued)

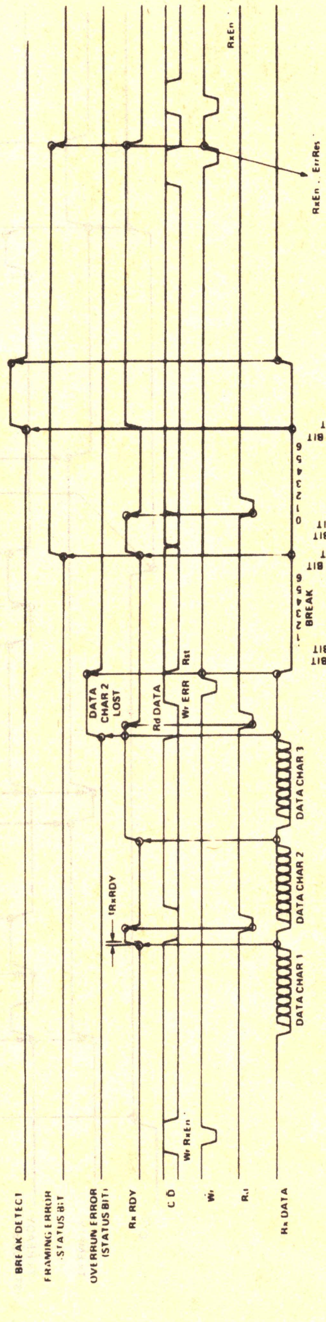
TRANSMITTER CONTROL AND FLAG TIMING (ASYNC MODE)



Example Format = 7 Bit Character With Parity & 2 Stop Bits.

205222-29

RECEIVER CONTROL AND FLAG TIMING (ASYNC MODE)

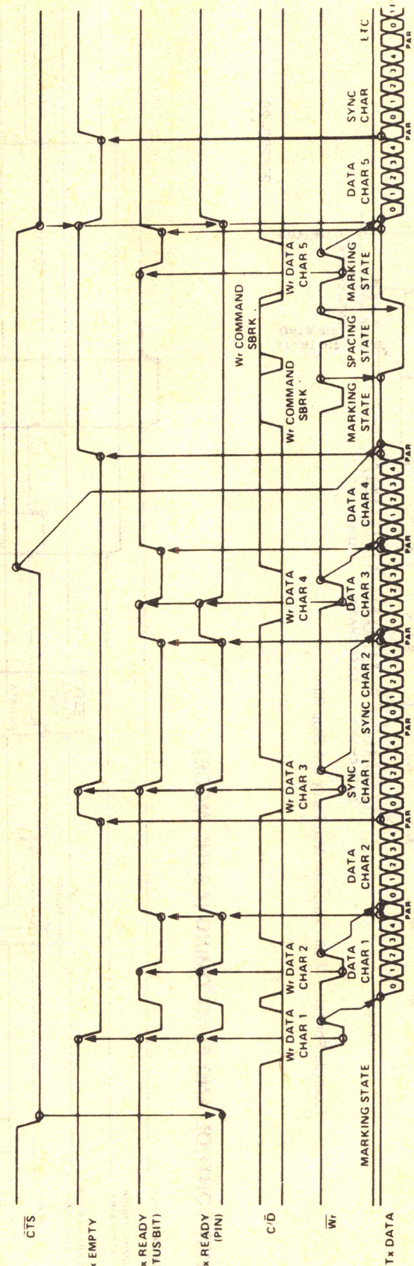


Example Format = 7 Bit Character With Parity & 2 Stop Bits

205222-30

WAVEFORMS (Continued)

TRANSMITTER CONTROL AND FLAG TIMING (SYNC MODE)

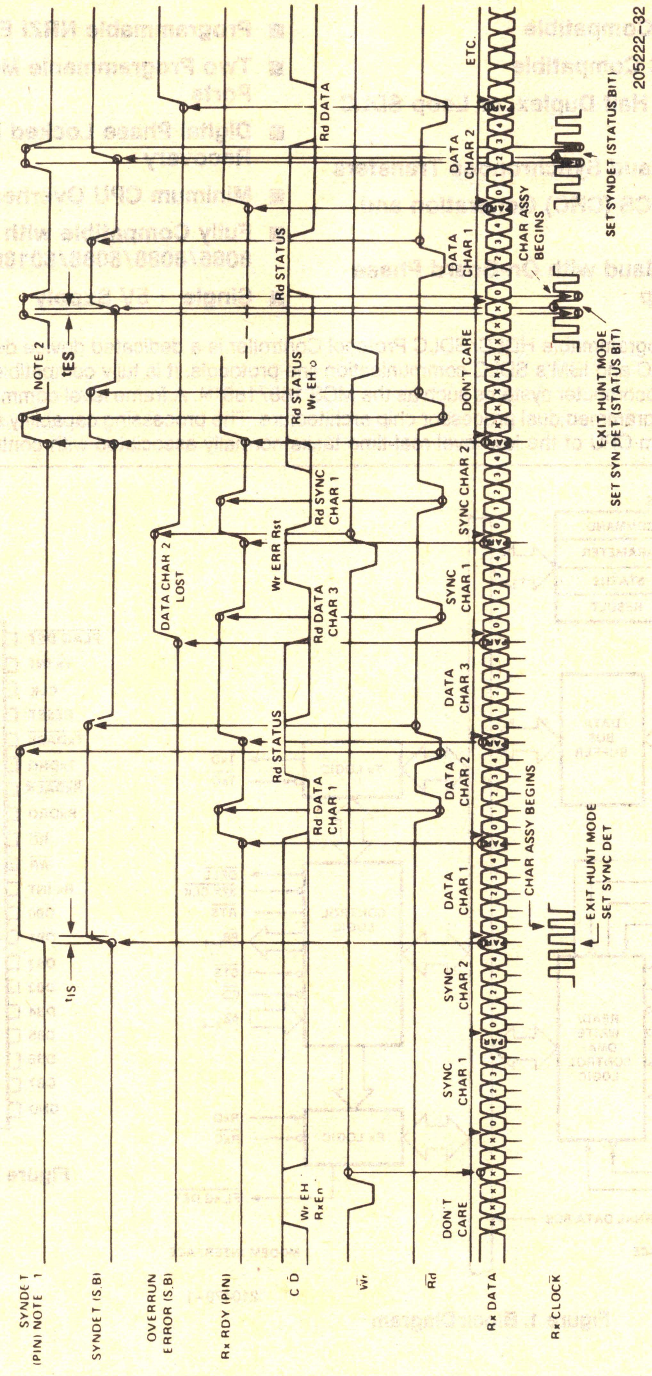


205222-31

Example Format = 5 Bit Character With Parity, 2 Sync Characters.

WAVEFORMS (Continued)

RECEIVER CONTROL AND FLAG TIMING (SYNC MODE)



NOTES:
1. Internal Sync, 2 Sync Characters, 5 Bits With Parity.
2. External Sync, 5 Bits, With Parity.

8273 PROGRAMMABLE HDLC/SDLC PROTOCOL CONTROLLER

- **CCITT X.25 Compatible**
- **HDLC/SDLC Compatible**
- **Full Duplex, Half Duplex, or Loop SDLC Operation**
- **Up to 64K Baud Synchronous Transfers**
- **Automatic FCS (CRC) Generation and Checking**
- **Up to 9.6K Baud with On-Board Phase Locked Loop**
- **Programmable NRZI Encode/Decode**
- **Two Programmable Modem Control Ports**
- **Digital Phase Locked Loop Clock Recovery**
- **Minimum CPU Overhead**
- **Fully Compatible with 8048/8080/8085/8088/8086/80188/80186 CPUs**
- **Single +5V Supply**

The Intel 8273 Programmable HDLC/SDLC Protocol Controller is a dedicated device designed to support the ISO/CCITT's HDLC and IBM's SDLC communication line protocols. It is fully compatible with Intel's new high performance microcomputer systems such as the MCS 188/186™. A frame level command set is achieved by a unique microprogrammed dual processor chip architecture. The processing capability supported by the 8273 relieves the system CPU of the low level real-time tasks normally associated with controllers.

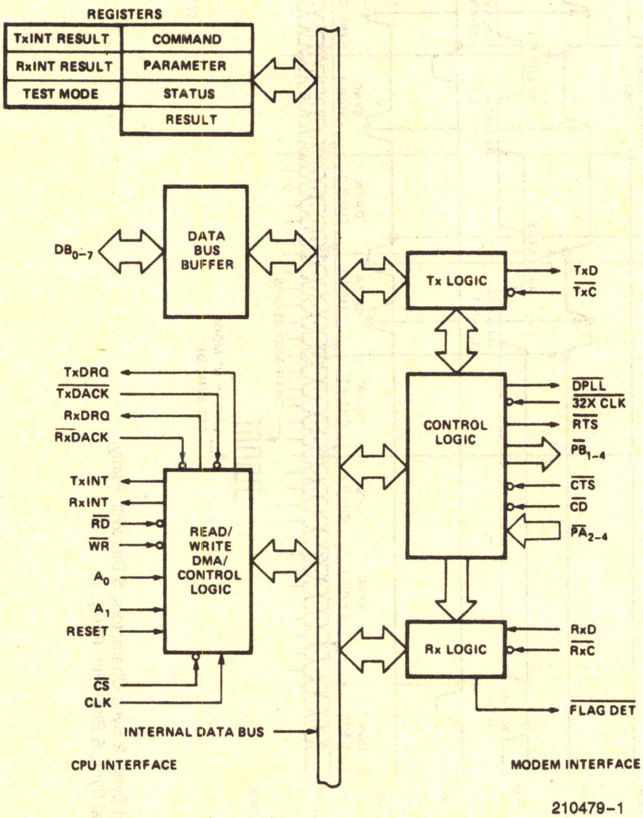


Figure 1. Block Diagram

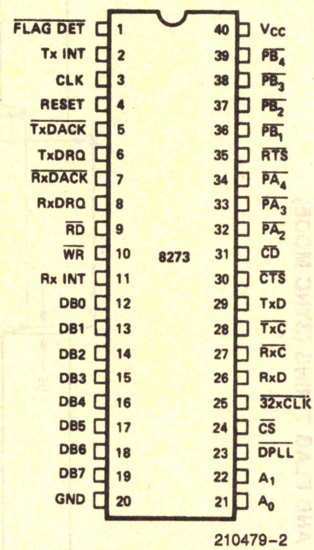


Figure 2. Configuration

November 1986
Order Number: 210479-003

A BRIEF DESCRIPTION OF HDLC/SDLC PROTOCOLS

General

The High Level Data Link Control (HDLC) is a standard communication link protocol established by International Standards Organization (ISO). HDLC is the discipline used to implement ISO X.25 packet switching systems.

The Synchronous Data Link Control (SDLC) is an IBM communication link protocol used to implement the System Network Architecture (SNA). Both the protocols are bit oriented, code independent, and ideal for full duplex communication. Some common applications include terminal to terminal, terminal to CPU, CPU to CPU, satellite communication, packet switching and other high speed data links. In systems which require expensive cabling and interconnect hardware, any of the two protocols could be used to simplify interfacing (by going serial), thereby reducing interconnect hardware costs. Since both the protocols are speed independent, reducing interconnect hardware could become an important application.

Network

In both the HDLC and SDLC line protocols, according to a pre-assigned hierarchy, a PRIMARY (Control) STATION controls the overall network (data link) and issues commands to the SECONDARY (Slave) STATIONS. The latter comply with instructions and respond by sending appropriate RESPONSES. Whenever a transmitting station must end transmission prematurely it sends an ABORT character. Upon detecting an abort character, a receiving station ignores the transmission block called a FRAME. Time fill between frames can be accomplished by transmitting either continuous frame preambles called FLAGS or an abort character. A time fill within a frame is not permitted. Whenever a station receives a string of more than fifteen consecutive ones, the station goes into an IDLE state.

Frames

A single communication element is called a FRAME which can be used for both Link Control and data transfer purposes. The elements of a frame are the beginning eight bit FLAG (F) consisting of one zero, six ones, and a zero, an eight bit ADDRESS FIELD (A), an eight bit CONTROL FIELD (C), a variable (N-bit) INFORMATION FIELD (I), a sixteen bit FRAME CHECK SEQUENCE (FCS), and an eight bit end FLAG (F), having the same bit pattern as the beginning flag. In HDLC the Address (A) and Control (C) bytes are extendable. The HDLC and the SDLC use three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

Frame Characteristics

An important characteristic of a frame is that is contents are made code transparent by use of a zero bit insertion and deletion technique. Thus, the user can adopt any format or code suitable for his system—it may even be a computer word length or a "memory dump". The frame is bit oriented that is, bits, not characters in each field, have specific meanings. The Frame Check Sequence (FCS) is an error detection scheme similar to the Cyclic Redundancy Checkword (CRC) widely used in magnetic disk storage devices. The Command and Response information frames contain sequence numbers in the control fields identifying the sent and received frames. The sequence numbers are used in Error Recovery Procedures (ERP) and as implicit acknowledgement of frame communication, enhancing the true full-duplex nature of the HDLC/SDLC protocols.

In contrast, BISYNC is basically half-duplex (two way alternate) because of necessity to transmit immediate acknowledgement frames. HDLC/SDLC therefore saves propagation delay times and have a potential of twice the throughput rate of BISYNC.

It is possible to use HDLC or SDLC over half duplex lines but there is a corresponding loss in throughput because both are primarily designed for full-duplex communication. As in any synchronous system, the bit rate is determined by the clock bits supplied by the modem, protocols themselves are speed independent.

A byproduct of the use of zero-bit insertion-deletion technique is the non-return-to-zero invert (NRZI) data transmission/reception compatibility. The latter allows HDLC/SDLC protocols to be used with asynchronous data communication hardware in which the clocks are derived from the NRZI encoded data.

References

IBM Synchronous Data Link Control General Information, IBM, GA27-3093-1.

Standard Network Access Protocol Specification, DATAPAC, Trans-Canada Telephone System CCG111

Recommendation X.25 ISO/CCITT March 2, 1976.

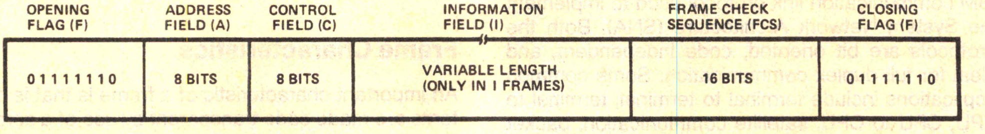
IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA 27-3098-0

Guidebook to Data Communciations, Training Manual, Hewlett-Packard 5955-1715

IBM Introduction to Teleprocessing, IBM, GC 20-8095-02

System Network Architecture, Technical Overview, IBM, GA 27-3102

System Network Architecture Format and Protocol, IBM GA 27-3112



210479-37

Figure 3. Frame Format

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	40		**POWER SUPPLY:** +5V Supply.
GND	20		**GROUND:** Ground.
RESET	4	I	**RESET:** A high signal on this pin will force the 8273 to an idle state. The 8273 will remain idle until a command is issued by the CPU. The modem interface output signals are forced high. Reset must be true for a minimum of 10 TCY.
$\overline{CS}$	24	I	**CHIP SELECT:** The RD and WR inputs are enabled by the chip select input.
DB_0–DB_7	12–19	I/O	**DATA BUS:** The Data Bus lines are bidirectional three-state lines which interface with the system Data Bus.
$\overline{WR}$	10	I	**WRITE INPUT:** The Write signal is used to control the transfer of either a command or data from CPU to the 8273.
$\overline{RD}$	9	I	**READ INPUT:** The Read signal is used to control the transfer of either a data byte or a status word from the 8273 to the CPU.
TxINT	2	O	**TRANSMITTER INTERRUPT:** The Transmitter interrupt signal indicates that the transmitter logic requires service.
RxINT	11	O	**RECEIVER INTERRUPT:** The Receiver interrupt signal indicates that the Receiver logic requires service.
TxDRQ	6	O	**TRANSMITTER DATA REQUEST:** Requests a transfer of data between memory and the 8273 for a transmit operation.
RxRDQ	8	O	**RECEIVER DMA REQUEST:** Requests a transfer of data between the 8273 and memory for a receive operation.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{\text{TXDACK}}$	5	I	**TRANSMITTER DMA ACKNOWLEDGE:** The Transmitter DMA acknowledge signal notifies the 8273 that the TxDMA cycle has been granted.
$\overline{\text{RxDACK}}$	7	I	**RECEIVER DMA ACKNOWLEDGE:** The Receiver DMA acknowledge signal notifies the 8273 that the RxDMA cycle has been granted.
A_0–A_1	21–22	I	**ADDRESS:** These two lines are CPU Interface Register Select lines.
TxD	29	O	**TRANSMITTER DATA:** This line transmits the serial data to the communication channel.
$\overline{\text{TxC}}$	28	I	**TRANSMITTER CLOCK:** The transmitter clock is used to synchronize the transmit data.
RxD	26	I	**RECEIVER DATA:** This line receives serial data from the communication channel.
$\overline{\text{RxC}}$	27	I	**RECEIVER CLOCK:** The Receiver Clock is used to synchronize the receive data.
$\overline{\text{32X CLK}}$	25	I	**32X CLOCK:** The 32X clock is used to provide clock recovery when an asynchronous modem is used. In loop configuration the loop station can run without an accurate 1X clock by using the 32X CLK in conjunction with the DPLL output. (This pin must be grounded when not used.)
$\overline{\text{DPLL}}$	23	O	**DIGITAL PHASE LOCKED LOOP:** Digital Phase Locked Loop output can be tied to RxC and/or TxC when 1X clock is not available. DPLL is used with 32X CLK.
$\overline{\text{FLAG DET}}$	1	O	**FLAG DETECT:** Flag Detect signals that a flag (01111110) has been received by an active receiver.
$\overline{\text{RTS}}$	35	O	**REQUEST TO SEND:** Request to Send signals that the 8273 is ready to transmit data.
$\overline{\text{CTS}}$	30	I	**CLEAR TO SEND:** Clear to Send signals that the modem is ready to accept data from the 8273.
$\overline{\text{CD}}$	31	I	**CARRIER DETECT:** Carrier Detect signals that the line transmission has started and the 8273 may begin to sample data on RxD line.
$\overline{\text{PA}}_{2-4}$	32–34	I	**GENERAL PURPOSE INPUT PORTS:** The logic levels on these lines can be Read by the CPU through the Data Bus Buffer.
$\overline{\text{PB}}_{1-4}$	36–39	O	**GENERAL PURPOSE OUTPUT PORTS:** The CPU can write these output lines through Data Bus Buffer.
CLK	3	I	**CLOCK:** A square wave TTL clock.

FUNCTIONAL DESCRIPTION

General

The Intel 8273 HDLC/SDLC controller is a microcomputer peripheral device which supports the International Standards Organization (ISO) High Level Data Link Control (HDLC), and IBM Synchronous Data Link Control (SDLC) communications protocols. This controller minimizes CPU software by supporting a comprehensive frame-level instruction set and by hardware implementation of the low level tasks associated with frame assembly/disassembly and data integrity. The 8273 can be used in either synchronous or asynchronous applications.

In asynchronous applications the data can be programmed to be encoded/decoded in NRZI code. The clock is derived from the NRZI data using a digital phase locked loop. The data transparency is achieved by using a zero-bit insertion/deletion technique. The frames are automatically checked for errors during reception by verifying the Frame Check Sequence (FCS); the FCS is automatically generated and appended before the final flag in transmit. The 8273 recognizes and can generate flags (01111110) Abort, Idle, and GA (EOP) characters.

The 8273 can assume either a primary (control) or a secondary (slave) role. It can therefore be readily implemented in an SDLC loop configuration as typified by the IBM 3650 Retail Store System by programming the 8273 into a one-bit delay mode. In such a configuration, a two wire pair can be effectively used for data transfer between controllers and loop stations. The digital phase locked loop output pin can be used by the loop station without the presence of an accurate Tx clock.

CPU Interface

The CPU interface is optimized for the MCS-80/85™ bus with an 8257 DMA controller. However, the interface is flexible, and allows either DMA or non-DMA data transfers, interrupt or non-interrupt driven. It further allows maximum line utilization by providing early interrupt mechanism for buffered (only the information field can be transferred to memory) Tx command overlapping. It also provides separate Rx and Tx interrupt output channels for efficient operation. The 8273 keeps the interrupt request active until all the associated interrupt results have been read.

The CPU utilizes the CPU interface to specify commands and transfer data. It consists of seven registers addressed via $\overline{CIA}$, A_1, A_0, $\overline{RD}$ and $\overline{WR}$ signals and two independent data registers for receive data and transmit data. A_1, A_0 are generally derived from two low order bits of the address bus. If an 8080 based CPU is utilized, the $\overline{RD}$ and $\overline{WR}$ signals may be driven by the 8228 $\overline{I/OR}$ and $\overline{I/OW}$. The table shows the seven register select decoding:

A_1	A_0	TxDACK	RxDACK	CS	RD	WR	Register
0	0	1	1	0	1	0	Command
0	0	1	1	0	0	1	Status
0	1	1	1	0	1	0	Parameter
0	1	1	1	0	0	1	Result
1	0	1	1	0	1	0	Reset
1	0	1	1	0	0	1	TxINT Result
1	1	1	1	0	1	0	—
1	1	1	1	0	0	1	RxINT Result
X	X	0	1	1	1	0	Transmit Data
X	X	1	0	1	0	1	Receive Data

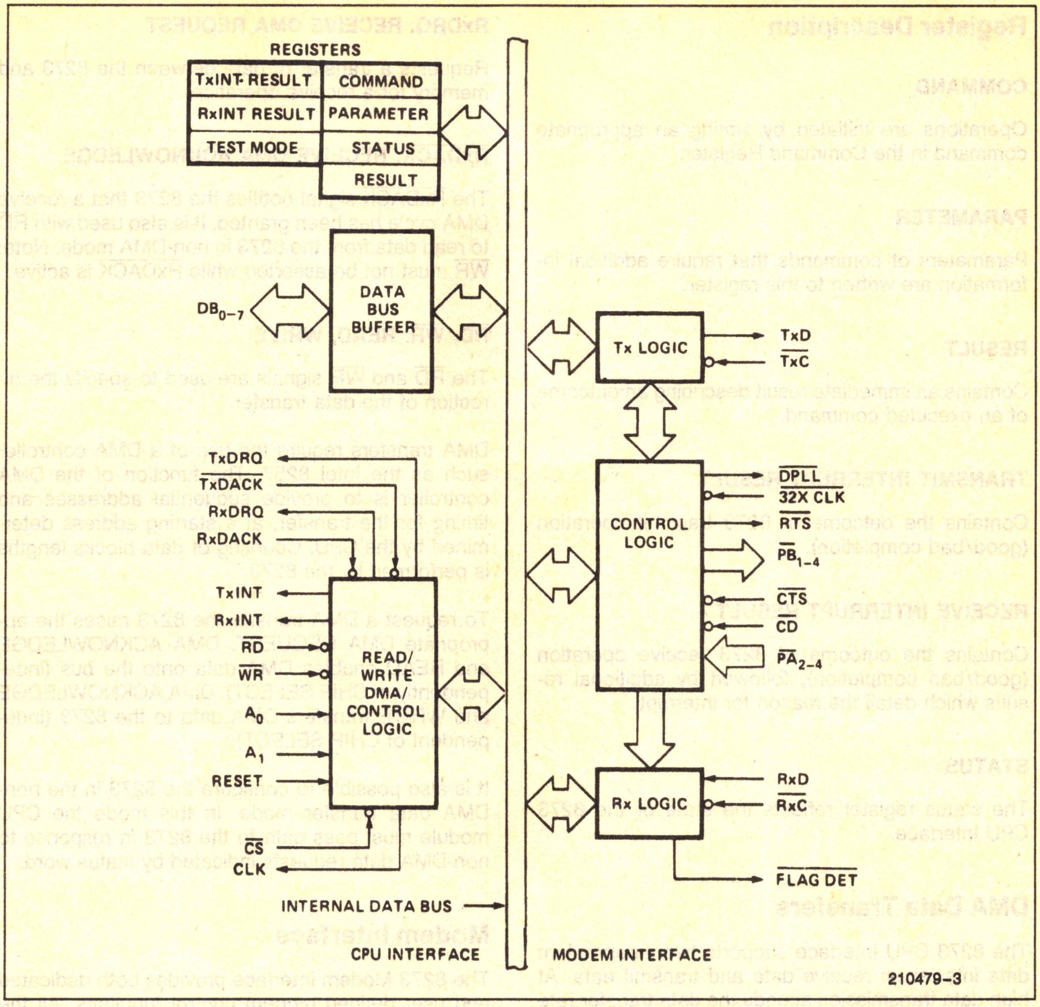


Figure 4. 8273 Block Diagram Showing CPU Interface Functions

Register Description

COMMAND

Operations are initiated by writing an appropriate command in the Command Register.

PARAMETER

Parameters of commands that require additioal information are written to this register.

RESULT

Contains an immediate result describing an outcome of an executed command.

TRANSMIT INTERRUPT RESULT

Contains the outcome of 8273 transmit operation (good/bad completion).

RECEIVE INTERRUPT RESULT

Contains the outcome of 8273 receive operation (good/bad completion), followed by additional results which detail the reason for interrupt.

STATUS

The status register reflects the state of the 8273 CPU Interface.

DMA Data Transfers

The 8273 CPU interface supports two independent data interfaces: receive data and transmit data. At high data transmission speeds the data transfer rate of the 8273 is great enough to justify the use of direct memory access (DMA) for the data transfers. When the 8273 is configured in DMA mode, the elements of the DMA interfaces are:

TxDRQ: TRANSMIT DMA REQUEST

Requests a transfer of data between memory and the 8273 for a transmit operation.

$\overline{\text{TxDACK}}$: TRANSMIT DMA ACKNOWLEDGE

The $\overline{\text{TxDACK}}$ signal notifies the 8273 that a transmit DMA cycle has been granted. It is also used with $\overline{\text{WR}}$ to transfer data to the 8273 in non-DMA mode. Note: $\overline{\text{RD}}$ must not be asserted while TxDACK is active.

RxDRQ: RECEIVE DMA REQUEST

Requests a transfer of data between the 8273 and memory for a receive operation.

$\overline{\text{RxDACK}}$: RECEIVE DMA ACKNOWLEDGE

The $\overline{\text{RxDACK}}$ signal notifies the 8273 that a receive DMA cycle has been granted. It is also used with $\overline{\text{RD}}$ to read data from the 8273 in non-DMA mode. Note: $\overline{\text{WR}}$ must not be asserted while $\overline{\text{RxDACK}}$ is active.

$\overline{\text{RD}}$, $\overline{\text{WR}}$: READ, WRITE

The $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals are used to specify the direction of the data transfer.

DMA transfers require the use of a DMA controller such as the Intel 8257. The function of the DMA controller is to provide sequential addresses and timing for the transfer, at a starting address determined by the CPU. Counting of data blocks lengths is performed by the 8273.

To request a DMA transfer the 8273 raises the appropriate DMA REQUEST. DMA ACKNOWLEDGE and READ enables DMA data onto the bus (independently of CHIP SELECT). DMA ACKNOWLEDGE and WRITE transfers DMA data to the 8273 (independent of CHIP SELECT).

It is also possible to configure the 8273 in the non-DMA data transfer mode. In this mode the CPU module must pass data to the 8273 in response to non-DMA data requests indicated by status word.

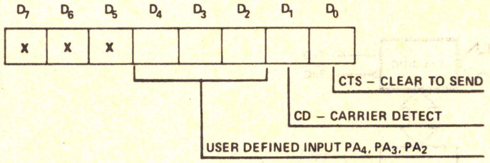
Modem Interface

The 8273 Modem interface provides both dedicated and user defined modem control functions. All the control signals are active low so that EIA RS-232C inverting drivers (MC 1488) and inverting receivers (MC 1489) may be used to interface to standard modems. For asynchronous operation, this interface supports programmable NRZI data encode/decode, a digital phase locked loop for efficient clock extraction from NRZI data, and modem control ports with automatic $\overline{\text{CTS}}$, $\overline{\text{CD}}$ monitoring and $\overline{\text{RTS}}$ generation. This interface also allows the 8273 to operate in PRE-FRAME SYNC mode in which the 8273 prefixes 16 transitions to a frame to synchronize idle lines before transmission of the first flag.

It should be noted that all the 8273 port operations deal with logical values, for instance, bit D0 of Port A will be a one when $\overline{\text{CTS}}$ (Pin 30) is a physical zero (logical one).

PORT A — INPUT PORT

During operation, the 8273 interrogates input pins $\overline{\text{CTS}}$ (Clear to Send) and $\overline{\text{CD}}$ (Carrier Detect). $\overline{\text{CTS}}$ is used to condition the start of a transmission. If during transmission $\overline{\text{CTS}}$ is lost the 8273 generates an interrupt. During reception, if $\overline{\text{CD}}$ is lost, the 8273 generates an interrupt.

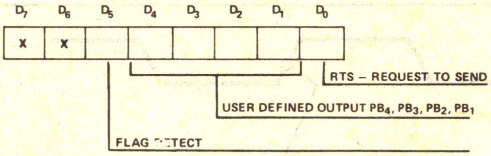


210479–38

The user defined input bits correspond to the 8273 PA_4, PA_3 and PA_2 pins. The 8273 does not interrogate or manipulate these bits.

PORT B - OUTPUT PORT

During normal operation, if the CPU sets $\overline{\text{RTS}}$ active, the 8273 will not change this pin; however, if the CPU sets $\overline{\text{RTS}}$ inactive, the 8273 will activate it before each transmission and deactivate it one byte time after transmission. While the receiver is active the flag detect pin is pulsed each time a flag sequence is detected in the receive data stream. Following an 8273 reset, all pins of Port B are set to a high, inactive level.



210479–39

The user defined output bits correspond to the state of PB_4–PB_1 pins. The 8273 does not interrogate or manipulate these bits.

Serial Data Logic

The Serial data is synchronized by the user transmit ($\overline{\text{TxC}}$) and receive ($\overline{\text{RxC}}$) clocks. The leading edge of $\overline{\text{TxC}}$ generates new transmit data and the trailing edge of $\overline{\text{RxC}}$ is used to capture receive data. The NRZI encoding/decoding of the receive and transmit data is programmable.

The diagnostic features included in the Serial Data logic are programmable loop back of data and selectable clock for the receiver. In the loop-back mode, the data presented to the TxD pin is internally routed to the receive data input circuitry in place of the RxD pin, thus allowing a CPU to send a message to itself to verify operation of the 8273.

In the selectable clock diagnostic feature, when the data is looped back, the receiver may be presented incorrect sample timing by the external circuitry. The user may select to substitute the $\overline{\text{TxC}}$ pin for the $\overline{\text{RxC}}$ input on-chip so that the clock used to generate the loop back data is used to sample it. Since TxD is generated off the leading edge of $\overline{\text{TxC}}$ and RxD is sampled on the trailing edge, the selected clock allows bit synchronism.

ASYNCHRONOUS MODE INTERFACE

Although the 8273 is fully compatible with the HDLC/SDLC communication line protocols, which are primarily designed for sychronous communication, the 8273 can also be used in asynchronous applications by using this interface. The interface employs a digital phase locked loop (DPLL) for clock recovery from a receive data stream and programmable NRZI encoding and decoding of data. The use of NRZI coding with SDLC transmission guarantees that within a frame, data transitions will occur at least every five bit times—the longest sequence of ones which may be transmitted without zero-bit insertion. The DPLL should be used only when NRZI coding is used since the NRZI coding will transmit zero sequence as line transitions. The digital phase locked loop also facilitates full-duplex and half-duplex asynchronous implementation with, or without modems.

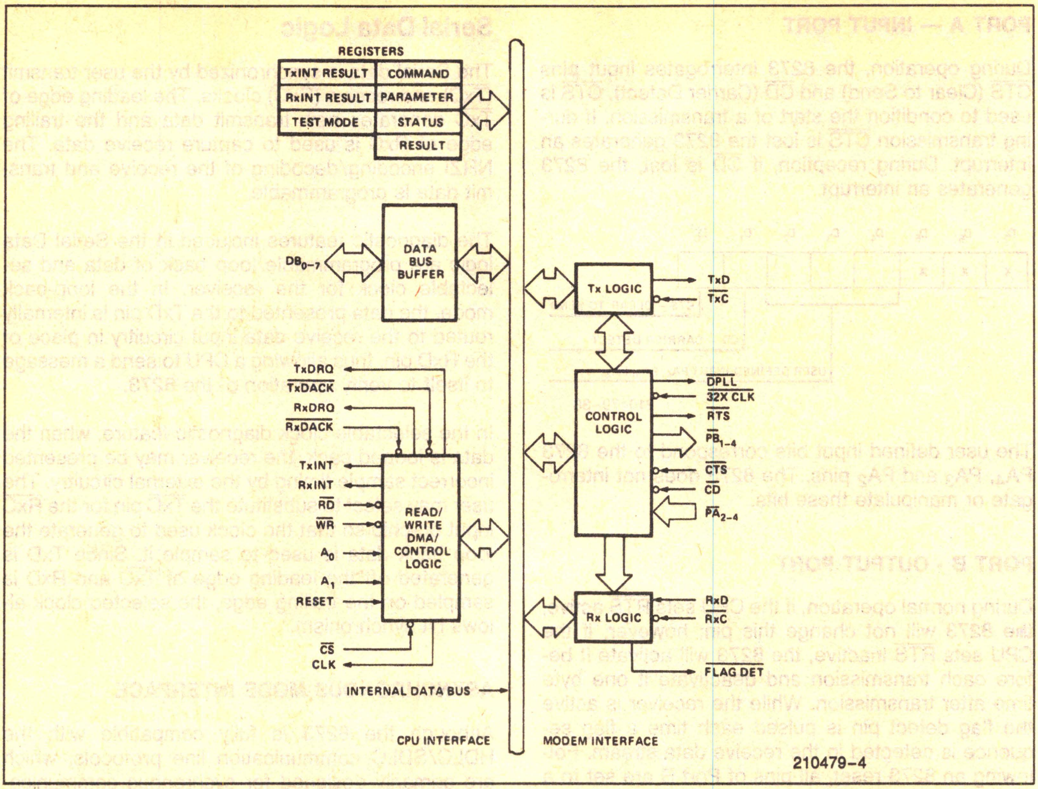


Figure 5. 8273 Block Diagram Showing Control Logic Functions

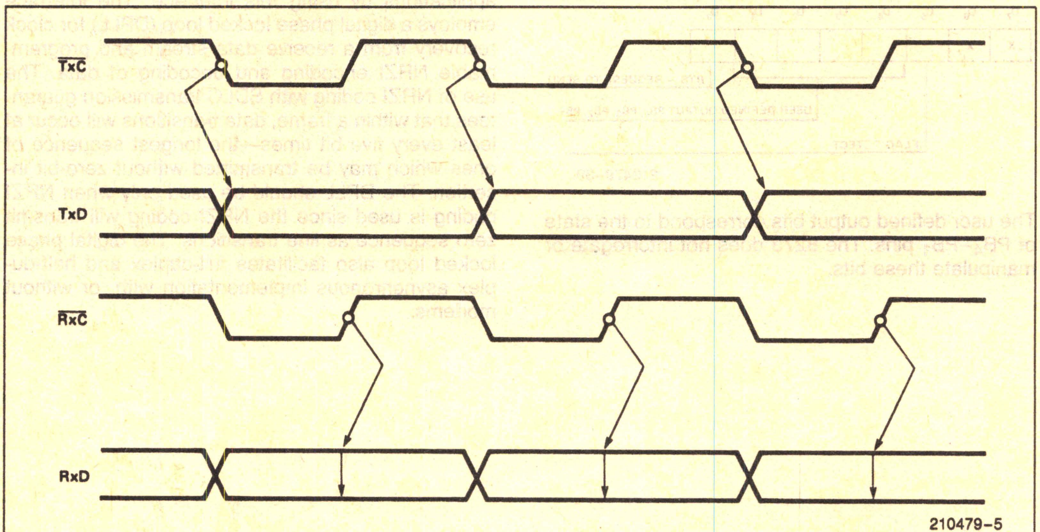


Figure 6. Transmit/Receive Timing

DIGITAL PHASE LOCKED LOOP

In asynchronous applications, the clock is derived from the receiver data stream by the use of the digital phase locked loop (DPLL). The DPLL requires a clock input at 32 times the required baud rate. The receive data (RxD) is sampled with this $\overline{32X\ CLK}$ and the 8273 DPLL supplies a sample pulse nomminally centered on the RxD bit cells. The DPLL has a built-in "stiffness" which reduces sensitivity to line noise and bit distortion. This is accomplished by making phase error adjustments in discrete increments. Since the nominal pulse is made to occur at 32 counts of the $\overline{32X\ CLK}$, these counts are subtracted or added to the nominal, depending upon which quadrant of the four error quadrants the data edge occurs in. For example if an RxD edge is detected in quadrant A1, it is apparent that the $\overline{DPLL}$ sample "A" was placed too close to the trailing edge of the data cell; sample "B" will then be placed at T = ($T_{nominal}$ - 2 counts) = 30 counts of the $\overline{32X\ CLK}$ to move the sample pulse "B" toward the nominal center of the next bit cell. A data edge occuring in quadrant B1 would cause a smaller adjustment of phase with T = 31 counts of the $\overline{32X\ CLK}$. Using this technique the $\overline{DPLL}$ pulse will converge to nominal bit center within 12 data bit times, worst case, with constant incoming RxD edges.

A method of attaining bit synchronism following a line idle is to use PRE-FRAME SYNC mode of transmission.

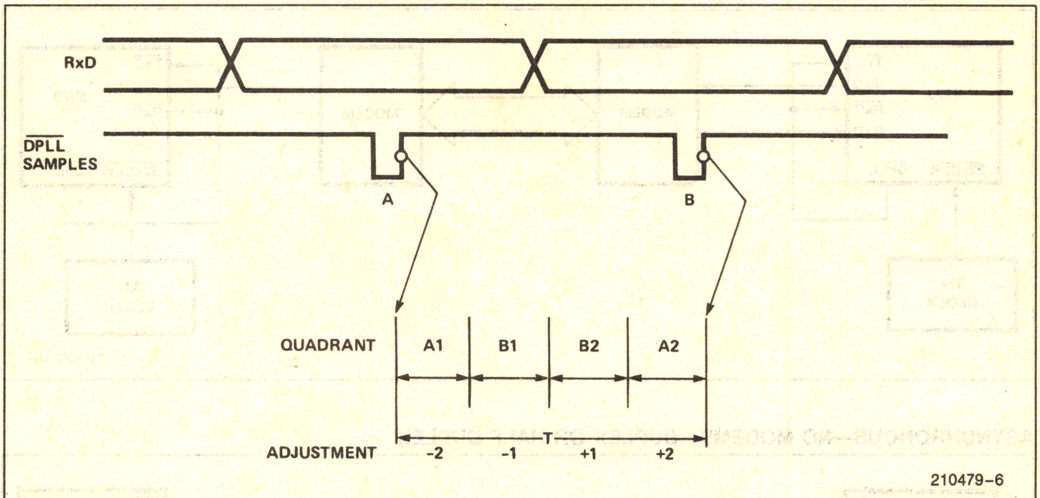
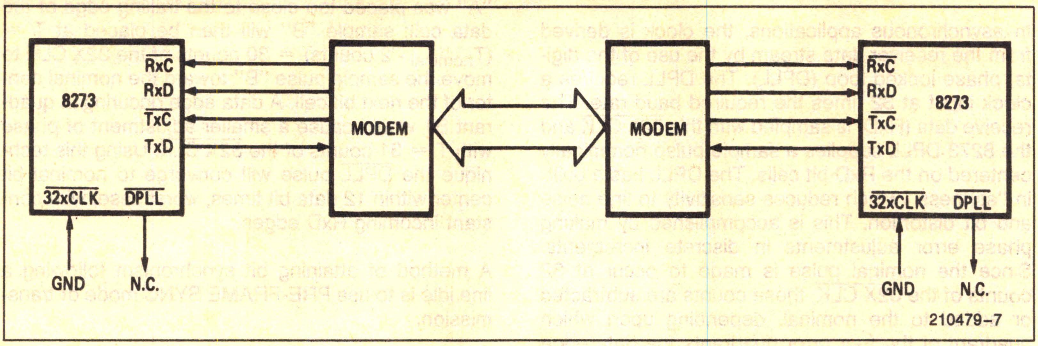


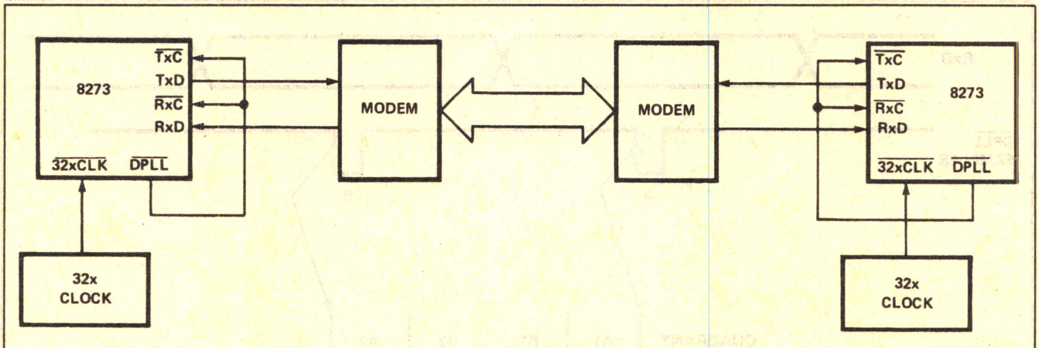
Figure 7. DPLL Sample Timing

SYNCHRONOUS MODEM—DUPLEX OR HALF DUPLEX OPERATION



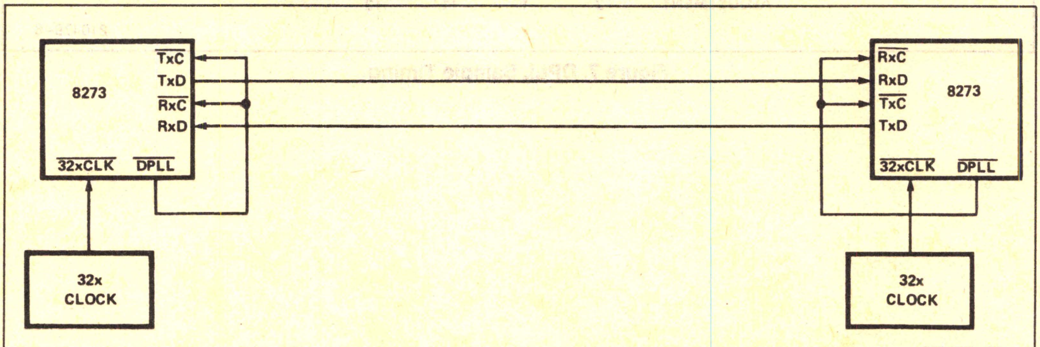
210479-7

ASYNCHRONOUS MODES—DUPLEX OR HALF DUPLEX OPERATION



210479-8

ASYNCHRONOUS—NO MODEMS—DUPLEX OR HALF DUPLEX



210479-33

SDLC LOOP

The DPLL simplifies the SDLC loop station implementation. In this application, each secondary station on a loop data link is a repeater set in one-bit delay mode. The signals sent out on the loop by the loop controller (primary station) are relayed from station to station then, back to the controller. Any secondary station finding its address in the A field captures the frame for action at that station. All received frames are relayed to the next station on the loop.

Loop stations are required to derive bit timing from the incoming NRZI data stream. The DPLL generates sample Rx clock timing for reception and uses the same clock to implement Tx clock timing.

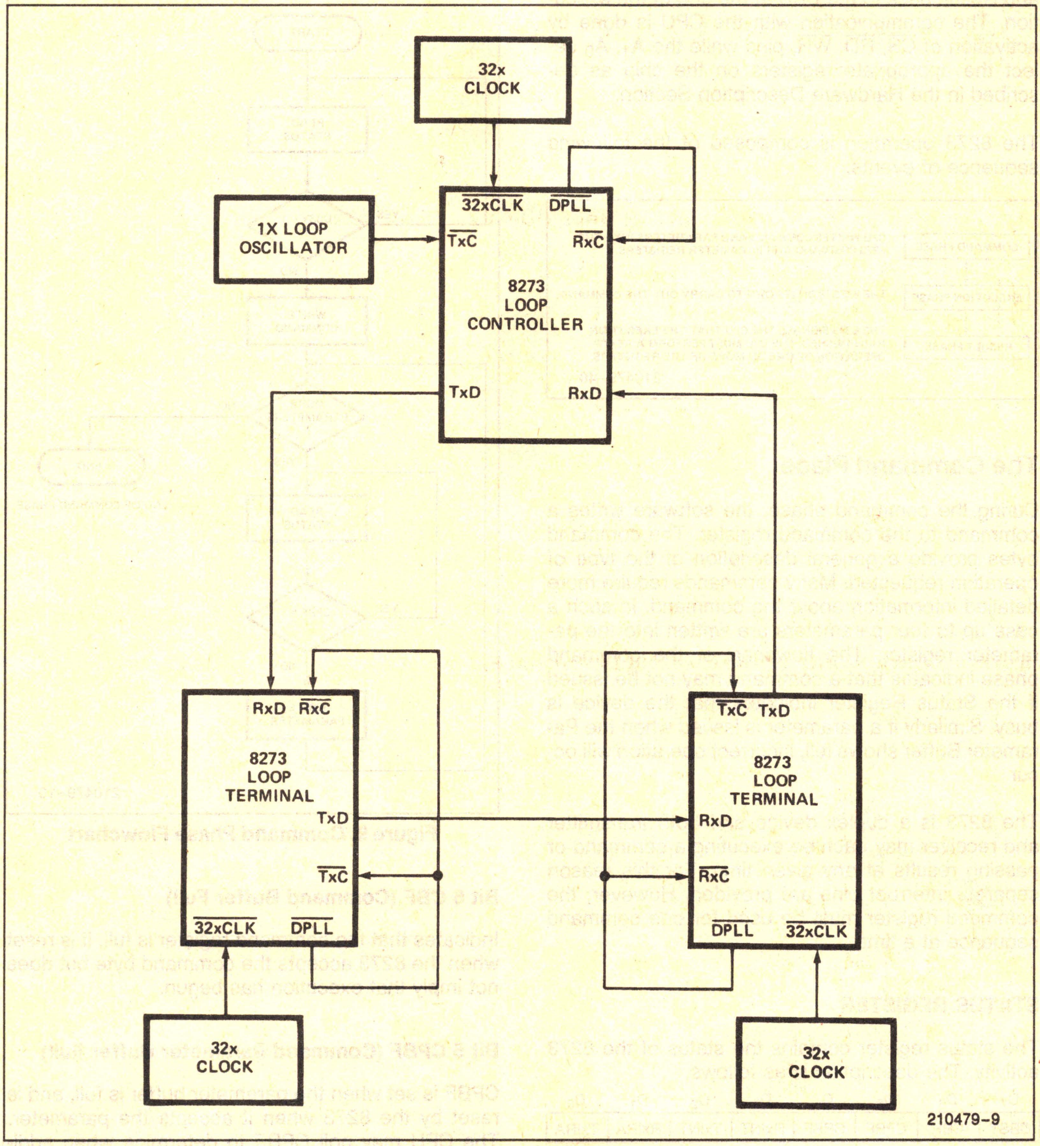
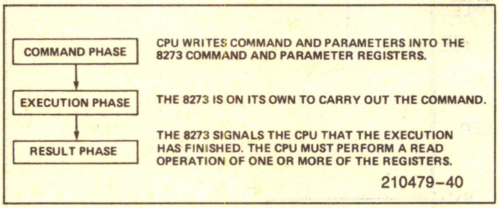


Figure 8. SDLC Loop Application

PRINCIPLES OF OPERATION

The 8273 is an intelligent peripheral controller which relieves the CPU of many of the rote tasks associated with constructing and receiving frames. It is fully compatible with the MCS-80/85™ system bus. As a peripheral device, it accepts commands from a CPU, executes these commands and provides an Interrupt and Result back to the CPU at the end of the execution. The communication with the CPU is done by activation of $\overline{CS}$, $\overline{RD}$, $\overline{WR}$, pins while the A_1, A_0 select the appropriate registers on the chip as described in the Hardware Description Section.

The 8273 operation is composed of the following sequence of events:



The Command Place

During the command phase, the software writes a command to the command register. The command bytes provide a general description of the type of operation requested. Many commands require more detailed information about the command. In such a case up to four parameters are written into the parameter register. The flowchart of the command phase indicates that a command may not be issued if the Status Register indicates that the device is busy. Similarly if a parameter is issued when the Parameter Buffer shows full, incorrect operation will occur.

The 8273 is a duplex device and both transmitter and receiver may each be executing a command or passing results at any given time. For this reason separate interrupt pins are provided. However, the command register must be used for one command sequence at a time.

STATUS REGISTER

The status register contains the status of the 8273 activity. The description is as follows.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CBSY	CBF	CPBF	CRBF	RxINT	TxINT	RxIRA	TxIRA

Bit 7 CBSY (Command Busy)

Indicates in-progress command, set for CPU poll when Command Register is full, reset upon command phase completion. It is improper to write a command when CBSY is set; it results in incorrect operation.

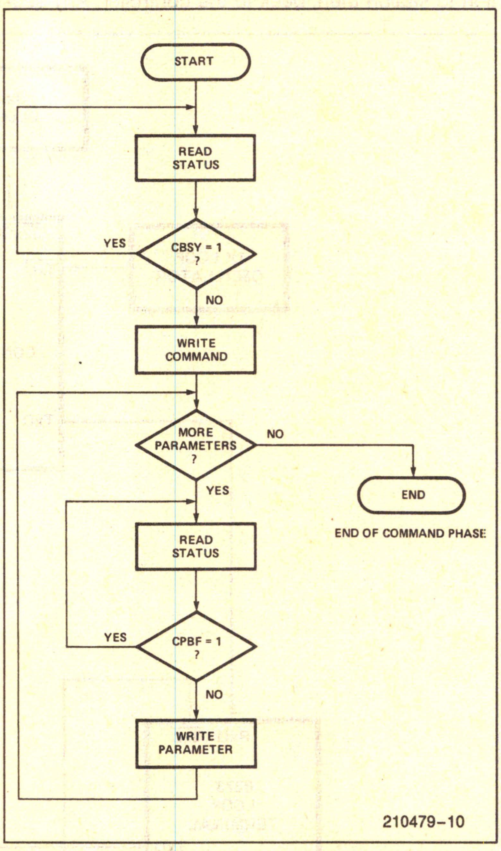


Figure 9. Command Phase Flowchart

Bit 6 CBF (Command Buffer Full)

Indicates that the command register is full, it is reset when the 8273 accepts the command byte but does not imply that execution has begun.

Bit 5 CPBF (Command Parameter Buffer Full)

CPBF is set when the parameter buffer is full, and is reset by the 8273 when it accepts the parameter. The CPU may poll CPBF to determine when additional parameters may be written.

Bit 4 CRBF (Command Result Buffer Full)

Indicates that an executed command immediate result is present in the Result Register. It is set by 8273 and reset when CPU reads the result.

Bit 3 RxINT (Receiver Interrupt)

RxINT indicates that the receiver requires CPU attention. It is identical to RxINT (pin 11) and is set by the 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has received a data byte from the 8273 in a Non-DMA data transfer.

Bit 2 TxINT (Transmitter Interrupt)

The TxINT indicates that the transmitter requires CPU attention. It is identical to TxINT (pin 2). It is set by 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has transferred transmit data byte to the 8273 in a Non-DMA transfer.

Bit 1 RxIRA (Receiver Interrupt Result Available)

The RxIRA is set by the 8273 when an interrupt result byte is placed in the RxINT register. It is reset after the CPU has read the RxINT register.

Bit 0 TxIRA (Transmitter Interrupt Result Available)

The TxIRA is set by the 8273 when an interrupt result byte is placed in the TxINT register. It is reset when the CPU has read the TxINT register.

THE EXECUTION PHASE

Upon accepting the last parameter, the 8273 enters into the Execution Phase. The execution phase may consist of a DMA or other activity, and may or may not require CPU intervention. The CPU intervention is eliminated in this phase if the system utilizes DMA for the data transfers, otherwise, for non-DMA data transfers, the CPU is interrupted by the 8273 via TxINT and RxINT pins, for each data byte request.

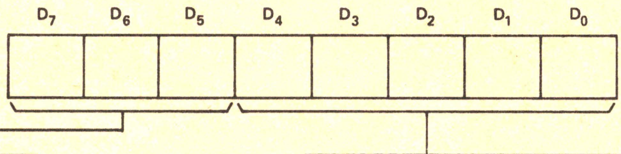
THE RESULT PHASE

During the result phase, the 8273 notifies the CPU of the execution outcome of a command. This phase is initiated by:

1. The successful completion of an operation
2. An error detected during an operation.

To facilitate quick network software decisions, two types of execution results are provided:

1. An Immediate Result
2. A Non-Immediate Result



D_7	D_6	D_5	
1	1	1	All 8 bits received
0	0	0	D_0 received
1	0	0	D_1-D_0 received
0	1	0	D_2-D_0 received
1	1	0	D_3-D_0 received
0	0	1	D_4-D_0 received
1	0	1	D_5-D_0 received
0	1	1	D_6-D_0 received

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Receiver Interrupt Result Code	Rx Status After INT
*	*	*	0	0	0	0	0	A_1 match or general receive	Active
*	*	*	0	0	0	0	1	A_2 match	Active
0	0	0	0	0	0	1	1	CRC error	Active
0	0	0	0	0	1	0	0	Abort detected	Active
0	0	0	0	0	1	0	1	Idle detect	Disabled
0	0	0	0	0	1	1	0	EOP detected	Disabled
0	0	0	0	0	1	1	1	Frame less than 32 bits	Active
0	0	0	0	1	0	0	0	DMA overrun detected	Disabled
0	0	0	0	1	0	0	1	Memory buffer overflow	Disabled
0	0	0	0	1	0	1	0	Carrier detect failure	Disabled
0	0	0	0	1	0	1	1	Receive interrupt overrun	Disabled

*Partial Byte Received

210479-11

Figure 10. Rx Interrupt Result Byte Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0					

	D_4	D_3	D_2	D_1	D_0
Early transmit interrupt	0	1	1	0	0
Frame transmit complete	0	1	1	0	1
DMA underrun	0	1	1	1	0
Clear to Send (CTS) error	0	1	1	1	1
Abort complete	1	0	0	0	0

210479–12

Figure 11. Tx Interrupt Result Byte Format

Immediate result is provided by the 8273 for commands such as Read Port A and Read Port B which have information ($\overline{CTS}$, $\overline{CD}$, $\overline{RTS}$, etc.) that the network software needs to make quick operational decisions.

A command which cannot provide an immediate result will generate an interrupt to signal the beginning of the Result phase. The immediate results are provided in the Result Register; all non-immediate results are available upon device interrupt, through Tx Interrupt Result Register TxI/R or Rx Interrupt Result Register RxI/R. The result may consist of a one-byte interrupt code indicating the condition for the interrupt and, if required, one or more bytes which detail the condition.

Tx and Rx Interrupt Result Registers

The Result Registers have a result code, the three high order bits D_7–D_5 of which are set to zero for all but the receive command. This command result contains a count that indicates the number of bits received in the last byte. If a partial byte is received, the high order bits of the last data byte are indeterminate.

All results indicated in the command summary must be read during the result phase.

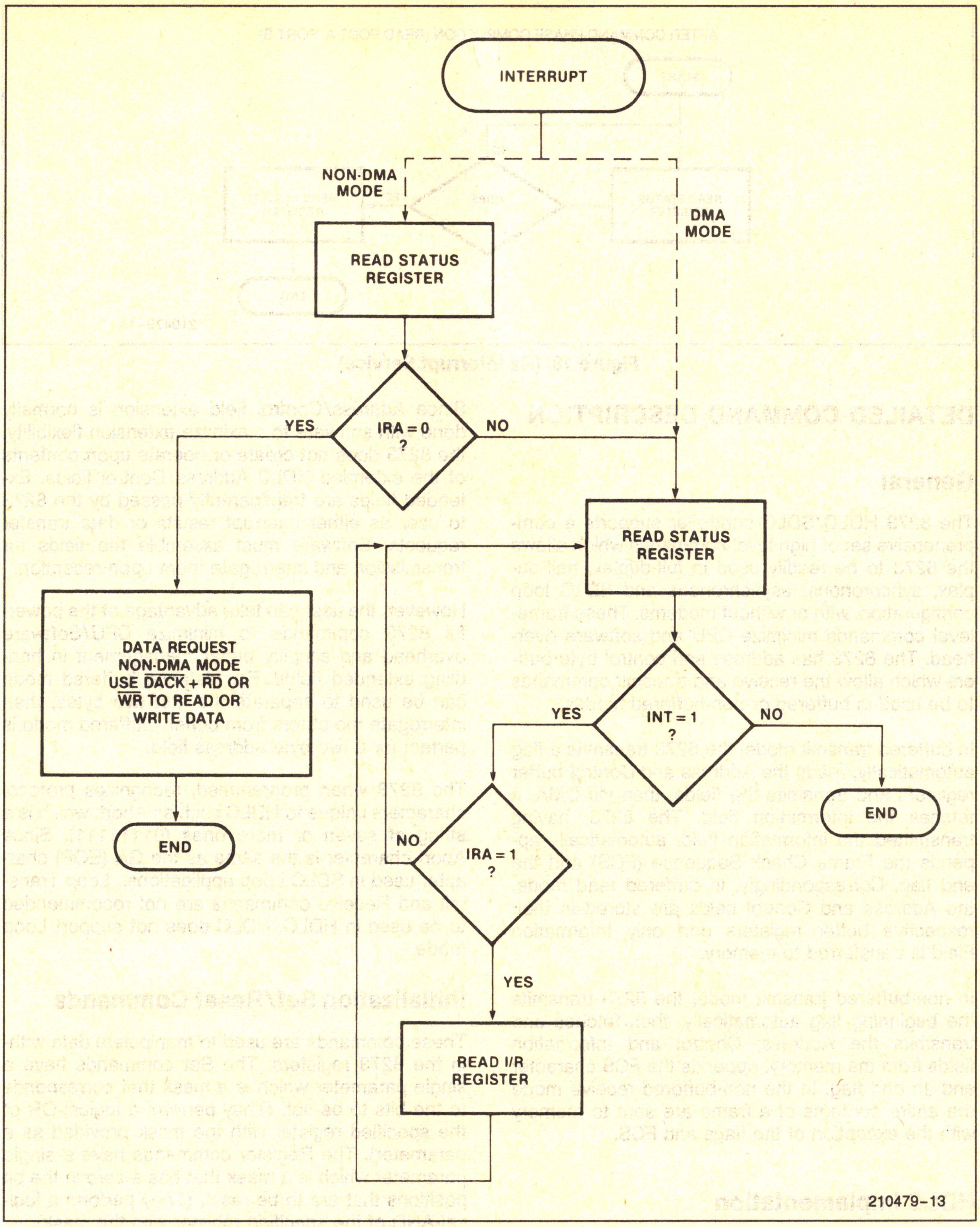


Figure 12. Result Phase Flowchart—Interrupt Results

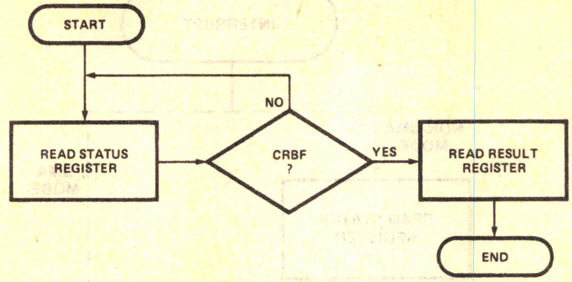


Figure 13. (Rx Interrupt Service)

DETAILED COMMAND DESCRIPTION

General

The 8273 HDLC/SDLC controller supports a comprehensive set of high level commands which allows the 8273 to be readily used in full-duplex, half-duplex, synchronous, asynchronous and SDLC loop configuration, with or without modems. These frame-level commands minimize CPU and software overhead. The 8273 has address and control byte buffers which allow the receive and transmit commands to be used in buffered or non-buffered modes.

In buffered transmit mode, the 8273 transmits a flag automatically, reads the Address and Control buffer registers and transmits the fields, then via DMA, it fetches the information field. The 8273, having transmitted the information field, automatically appends the Frame Check Sequence (FCS) and the end flag. Correspondingly, in buffered read mode, the Address and Control fields are stored in their respective buffer registers and only Information Field is transferred to memory.

In non-buffered transmit mode, the 8273 transmits the beginning flag automatically, then fetches and transmits the Address, Control and Information fields from the memory, appends the FCS character and an end flag. In the non-buffered receive mode the entire contents of a frame are sent to memory with the exception of the flags and FCS.

HDLC Implementation

HDLC Address and Control field are extendable. The extension is selected by setting the low order bit of the field to be extended to a one, a zero in the low order bit indicates the last byte of the respective field.

Since Address/Control field extension is normally done with software to maximize extension flexibility, the 8273 does not create or operate upon contents of the extended HDLC Address/Control fields. Extended fields are transparently passed by the 8273 to user as either interrupt results or data transfer requests. Software must assemble the fields for transmission and interrogate them upon reception.

However, the user can take advantage of the powerful 8273 commands to minimize CPU/Software overhead and simplify buffer management in handling extended fields. For instance buffered mode can be used to separate the first two bytes, then interrogate the others from buffer. Buffered mode is perfect for a two byte address field.

The 8273 when programmed, recognizes protocol characters unique to HDLC such as Abort, which is a string of seven or more ones (01111111). Since Abort character is the same as the GA (EOP) character used in SDLC Loop applications., Loop Transmit and Receive commands are not recommended to be used in HDLC. HDLC does not support Loop mode.

Initialization Set/Reset Commands

These commands are used to manipulate data within the 8273 registers. The Set commands have a single parameter which is a mask that corresponds to the bits to be set. (They perform a logical-OR of the specified register with the mask provided as a parameter). The Register commands have a single parameter which is a mask that has a zero in the bit positions that are to be reset. (They perform a logical-AND of the specified register with the mask).

SET ONE-BIT DELAY (CMD CODE A4)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	1	0	0	1	0	0
PAR:	0	1	1	0	0	0	0	0	0	0

When one bit delay is set, 8273 retransmits the received data stream one bit delayed. This mode is entered at a receiver character boundary, and should only be used by Loop Stations.

RESET ONE-BIT DELAY (CMD CODE 64)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	1	1	0	0	1	0	0
PAR:	0	1	0	1	1	1	1	1	1	1

The 8273 stops the one bit delayed retransmission mode.

SET DATA TRANSFER MODE (CMD CODE 97)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	0	1	0	1	1	1
PAR:	0	1	0	0	0	0	0	0	0	1

When the data transfer mode is set, the 8273 will interrupt when data bytes are required for transmission or are available from a receive. If a transmit interrupt occurs and the status indicates that there is no Transmit Result (TxIRA = 0), the interrupt is a transmit data request. If a receive interrupt occurs and the status indicates that there is no receive result (RxIRA = 0), the interrupt is a receive data request.

RESET DATA TRANSFER MODE (CMD CODE 57)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	1	0	1	0	1	1	1
PAR:	0	1	1	1	1	1	1	1	1	0

If the Data Transfer Mode is reset, the 8273 data transfers are performed through the DMA requests without interrupting the CPU.

SET OPERATING MODE (CMD CODE 91)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	0	1	0	0	0	1
PAR:	0	1	0	0						

1 = FLAG STREAM MODE
1 = PREFRAME SYNC MODE
1 = BUFFERED MODE
1 = EARLY INTERRUPT MODE
1 = EOP INTERRUPT MODE
1 = HDLC MODE

210479–34

RESET OPERATING MODE (CMD CODE 51)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	1	0	1	0	0	0	1
PAR:	0	1	1	1						

Any mode switches set in CMD code 91 can be reset using this command by placing zeros in the appropriate positions.

(D5) HDLC MODE

In HDLC mode, a bit sequence of seven ones (01111111) is interpreted as as an abort character. Otherwise, eight ones (011111111) signal an abort.

(D4) EOP INTERRUPT MODE

In EOP interrupt mode, an interrupt is generated whenever an EOP character (01111111) is detected by an active receiver. This mode is useful for the implementation of an SDLC loop controller in detecting the end of a message stream after a loop poll.

(D3) TRANSMITTER EARLY INTERRUPT MODE (Tx)

The early interrupt mode is specified to indicate when the 8273 should generate an end of frame interrupt. When set, an early interrupt is generated when the last data character has been passed to the 8273. If the user software responds with another transmit command before the final flag is sent, the final flag interrupt will not be generated and a new frame will immediately begin when the current frame is complete. This permits frames to be separated by a single flag. If no additional Tx commands are provided, a final interrupt will follow.

NOTE:
In buffered mode, if a supervisory frame (no Information) Transmit command is sent in response to an early Transmit Interrupt, the 8273 will repeatedly transmit the same supervisory frame with one flag in between, until a non-supervisory transmit is issued.

Early transmitter interrupt can be used in buffered mode by waiting for a transmit complete interrupt instead of early Transmit Interrupt before issuing a transmit frame command for a supervisory frame. See Figure 14.

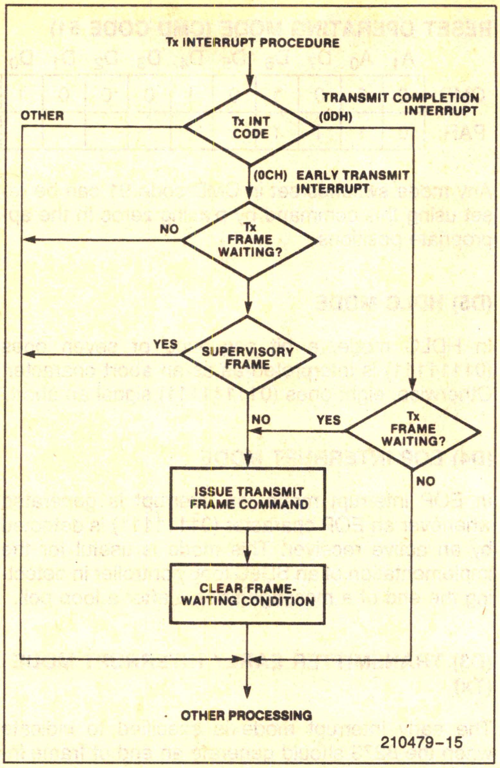


Figure 14

If this bit is zero, the interrupt will be generated only after the final flag has been transmitted.

(D2) BUFFERED MODE

If the buffered mode bit is set to a one, the first two bytes (normally the address (A) and control (C) fields) of a frame are buffered by the 8273. If this bit is a zero the address and control fields are passed to and from memory.

(D1) PREFRAME SYNC MODE

If this bit is set to a one the 8273 will transmit two characters before the first flag of a frame.

To guarantee sixteen line transitions, the 8273 sends two bytes of data $(00)_H$ if NRZI is set or data $(55)_H$ if NRZI is not set.

(D0) FLAG STREAM MODE

If this bit is set to a one, the following table outlines the operation of the transmitter.

Transmitter State	Action
Idle	Send Flags Immediately.
Transmit or Transmit Transparent Active	Send Flags After the Transmission Complete
Loop Transmit Active	Ignore Command.
1 Bit Delay Active	Ignore Command.

If this bit is reset to zero the following table outlines the operation of the transmitter

Transmitter State	Action
IDLE	Sends Idles on Next Character boundary.
Transmit or Transmit-Transparent Active	Send Idles after the Transmission is Complete.
Loop Transmit Active	Ignore Command.
1 Bit Delay Active	Ignore Command.

SET SERIAL I/O MODE (CMD CODE A0)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	1	0	0	0	0	0
PAR:	0	1	0	0	0	0	0			

1 = NRZ1 MODE
1 = TxC → RxC
1 = LOOP BACK TxD → RxD

210479-16

RESET SERIAL I/O MODE (CMD CODE 60)

This command allows bits set in CMD code A0 to be reset by placing zeros in the appropriate positions.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	1	1	0	0	0	0	0
PAR:	0	1	1	1	1	1	1			

(D2) LOOP BACK

If this bit is set to a one, the transmit data is internally routed to the receive data circuitry.

(D1) TxC → RxC

If this bit is set to a one, the transmit clock is internally routed to the receive clock circuitry. It is normally used with the loop back bit (D2).

(D0) NRZI MODE

If this bit is set to a one, NRZI encoding and decoding of transmit and receive data is provided. If this bit is a zero, the transmit and receive data is treated as a normal positive logic bit stream.

NRZI encoding specifies that a zero causes a change in the polarity of the transmitted signal and a one causes no polarity change. NRZI is used in all asynchronous operations. Refer to IBM document GA27-3093 for details.

Reset Device Command

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TMR:	1	0	0	0	0	0	0	0	0	1
TMR:	1	0	0	0	0	0	0	0	0	0

An 8273 reset command is executed by outputting a $(01)_H$ followed by $(00)_H$ to the reset register (TMR). See 8273 AC timing characteristics for Reset pulse specifications.

The reset command emulates the action of the reset pin.

1) The modem control signals are forced high (inactive level).
2) The 8273 status register flags are cleared.
3) Any commands in progress are terminated immediately.
4) The 8273 enters an idle state until the next command is issued.
5) The Serial I/O and Operating Mode registers are set to zero and DMA data register transfer mode is selected.
6) The device assumes a non-loop SDLC terminal role.

Receive Commands

The 8273 supports three receive commands: General Receive, Selective Receive, and Selective Loop Receive.

GENERAL RECEIVE (CMD CODE C0)

General receive is a receive mode in which frames are received regardless of the contents of the address field.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	0	0	0	0
PAR:	0	1	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)							

NOTES:

1. If buffered mode is specified, the R0, R1 receive frame length (result) is the number of data bytes received.
2. If non-buffered mode is specified, the R0, R1 receive frame length (result) is the number of data bytes received plus two (the count includes the address and control bytes).
3. The frame check sequence (FCS) is not transferred to memory.
4. Frames with less than 32 bits between flags are ignored (no interrupt generated) if the buffered mode is specified.
5. In the non-buffered mode an interrupt is generated when a less than 32 bit frame is received, since data transfer requests have occurred.
6. The 8273 receive is always disabled when an Idle is received after a valid frame. The CPU module must issue a receive command to re-enable the receiver.
7. The intervening ABORT character between a final flag and an IDLE does not generate an interrupt.
8. If an ABORT Character is not preceded by a flag and is followed by an IDLE, an interrupt will be generated for the ABORT followed by an IDLE interrupt one character time later. The reception of an ABORT will disable the receiver.

SELECTIVE RECEIVE (CMD CODE C1)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	0	0	0	1
PAR:	0	1	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)							
PAR:	0	1	RECEIVE FRAME ADDRESS MATCH FIELD ONE (A1)							
PAR:	0	1	RECEIVE FRAME ADDRESS MATCH FIELD TWO (A2)							

Selective receive is a receive mode in which frames are ignored unless the address field matches any one of two address fields given to the 8273 as parameters.

When selective receive is used in HDLC the 8273 looks at the first character, if extended, software must then decide if the message is for this unit.

SELECTIVE LOOP RECEIVE (CMD CODE C2)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	0	0	1	0
PAR:	0	0	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)							
PAR:	0	1	RECEIVE FRAME ADDRESS MATCH FIELD ONE (A1)							
PAR:	0	1	RECEIVE FRAME ADDRESS MATCH FIELD TWO (A2)							

Selective loop receive operates like selective receive except that the transmitter is placed in flag stream mode automatically after detecting an EOP (01111111) following a valid received frame. The one bit delay mode is also reset at the end of a selective loop receive.

RECEIVE DISABLE (CMD CODE 5)

Terminates an active receive command immediately.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	0	1	0	1
PAR:	NONE									

Transmit Commands

The 8273 supports three transmit commands: Transmit Frame, Loop Transmit, Transmit Transparent.

TRANSMIT FRAME (CMD CODE C8)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	0	0	0
PAR:	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							
PAR:	0	1	ADDRESS FIELD OF TRANSMIT FRAME (A)							
PAR:	0	1	CONTROL FIELD OF TRANSMIT FRAME (C)							

Transmits one frame including: initial flag, frame check sequence, and the final flag.

If the buffered mode is specified, the L0, L1, frame length provides as a parameter is the length of the information field and the address and control fields must be input.

In unbuffered mode the frame length provided must be the length of the information field plus two and the address and control fields must be the first two bytes of data. Thus only the frame length bytes are required as parameters.

LOOP TRANSMIT (CMD CODE CA)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	0	1	0
PAR:	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							
PAR:	0	1	ADDRESS FIELD OF TRANSMIT FRAME (A)							
PAR:	0	1	CONTROL FIELD OF TRANSMIT FRAME (C)							

Transmits one frame in the same manner as the transmit frame command except:

1) If the flag stream mode is not active transmission will begin after a received EOP has been converted to a flag.
2) If the flag stream mode is active transmission will begin at the next flag boundary for buffered mode or at the third flag boundary for non-buffered mode.
3) At the end of a loop transmit the one-bit delay mode is entered and the flag stream mode is reset.

TRANSMIT TRANSPARENT (CMD CODED C9)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	0	0	1
PAR:	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR:	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							

The 8273 will transmit a block of raw data without protocol, i.e., no zero bit insertion, flags, or frame check sequences.

Abort Transmit Commands

An abort command is supported for each type of transmit command. The abort commands are ignored if a transmit command is not in progress.

ABORT TRANSMIT FRAME (CMD CODE CC)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	1	0	0

PAR: NONE

After an abort character (eight contiguous ones) is transmitted, the transmitter reverts to sending flags or idles as a function of the flag stream mode specified.

ABORT LOOP TRANSMIT (CMD CODE CE)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	1	1	0

PAR: NONE

After a flag is transmitted the transmitter reverts to one bit delay mode.

ABORT TRANSMIT TRANSPARENT (CMD CODE CD)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	1	0	1

PAR: NONE

The transmitter reverts to sending flags or idles as a function of the flag stream mode specified.

Modem Control Commands

The modem control commands are used to manipulate the modem control ports.

When read Port A or Port B commands are executed the result of the command is returned in the result register. The Bit Set Port B command requires a parameter that is a mask that corresponds to the bits to be set. The Bit Reset Port B command requires a mask that has a zero in the bit positions that are to be reset.

READ PORT A (CMD CODE 22)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	0	1	0	0	0	1	0

PAR: NONE

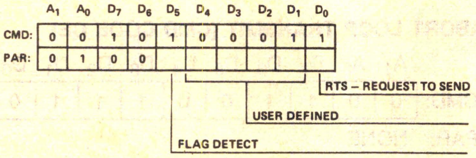
READ PORT B (CMD CODE 23)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	0	1	0	0	0	1	1

PAR: NONE

SET PORT B BITS (CMD CODE A3)

This command allows user defined Port B pins to be set.



210479–35

(D_5) FLAG DETECT

This bit can be used to set the flag detect pin. However, it will be reset when the next flag is detected.

(D_4–D_1) USER DEFINED OUTPUTS

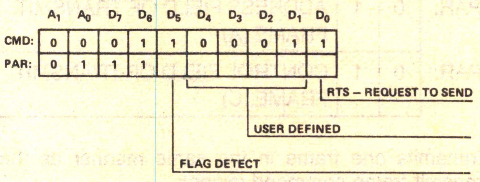
These bits correspond to the state of the PB_4–PB_1 output pins.

(D_0) REQUEST TO SEND

This is a dedicted 8273 modem control signal, and reflects the same logical state of RTS pin.

RESET PORT B BITS (CMD CODE 63)

This command allows Port B user defined bits to be reset.



210479–36

This command allows Port B (D_4–D_1) user defined bits to be reset. These bits correspond to Output Port pins (PB_4–PB_1).

8273 Command Summary

Command Description	Command HEX	Parameter	Results	Result Port	Completion Interrupt
Set One Bit Delay	A4	Set Mask	None	—	No
Reset One Bit Delay	64	Reset Mask	None	—	No
Set Data Transfer Mode	97	Set Mask	None	—	No
Reset Data Transfer Mode	57	Reset Mask	None	—	No
Set Operating Mode	91	Set Mask	None	—	No
Reset Operating Mode	51	Reset Mask	None	—	No
Set Serial I/O Mode	A0	Set Mask	None	—	No
Reset Serial I/O Mode	60	Reset Mask	None	—	No
General Receive	C0	B0, B1	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Selective Receive	C1	B0,B1,A1,A2	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Selective Loop Receive	C2	B0,B1,A1,A2	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Receive Disable	C5	None	None	—	No
Transmit Frame	C8	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Loop Transmit	CA	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Transmit Transparent	C9	L0,L1	TIC	TXI/R	Yes
Abort Transmit Frame	CC	None	TIC	TXI/R	Yes

8273 Command Summary (Continued)

Command Description	Command HEX	Parameter	Results	Result Port	Completion Interrupt
Abort Loop Transmit	CE	None	TIC	TXI/R	Yes
Abort Transmit Transparent	CD	None	TIC	TXI/R	Yes
Read Port A	22	None	Port Value	Result	No
Read Port B	23	None	Port Value	Result	No
Set Port B Bit	A3	Set Mask	None	—	No
Reset Port B Bit	63	Reset Mask	None	—	No

NOTES:
1. Issued only when in buffered mode.
2. Read as results only in buffered mode.

8273 Command Summary Key

B0— Least significant byte of the receiver buffer length.

B1— Most significant byte of the receive buffer length.

L0— Least significant byte of the Tx frame length.

L1— Most significant byte of the Tx frame length.

A1— Receive frame address match field one.

A2— Receive frame address match field two.

A— Address field of received frame. If non-buffered mode is specified, this result is not provided.

C— Control field of received frame. If non-buffered mode is specified this result is not provided.

RXI/R— Receive interrupt result register.

TXI/R— Transmit interrupt result register.

R0— Least significant byte of the length of the frame received.

R1— Most significant byte of the length of the frame received.

RIC— Receiver interrupt result code.

TIC— Transmitter interrupt result code.

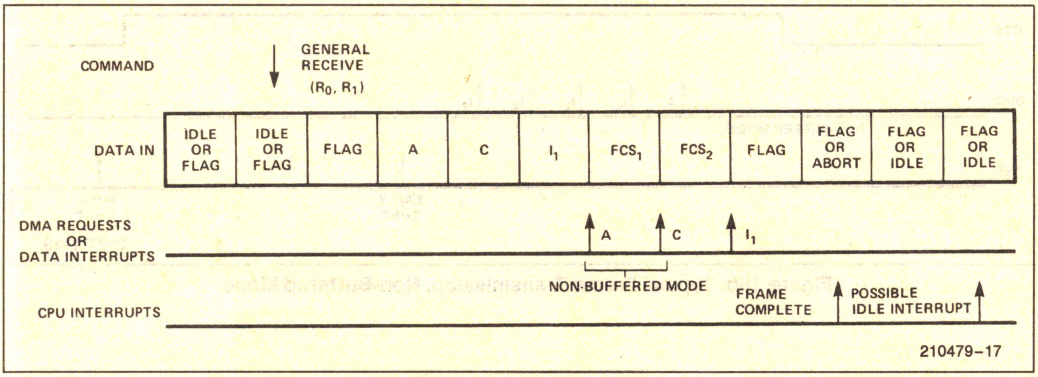


Figure 15. Typical Frame Reception

NOTE:
In order to ensure proper operation to the maximum baud rate, Receive commands or Read/Write Port commands should be written only when either the transmitter or the receiver is inactive. In full duplex systems, it is recommended that these commands be issued after servicing a transmitter interrupt but before a new transmit command is issued. When operating in full Duplex (active transmitter or receiver) with commands, the maximum data rate decreases to 49K Baud.

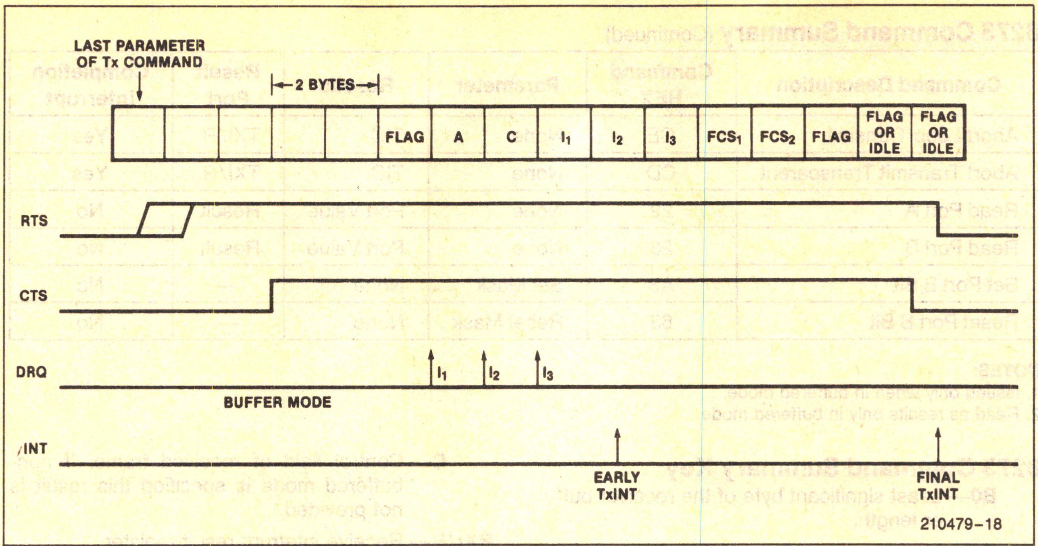


Figure 16a. Typical Frame Transmission, Buffered Mode

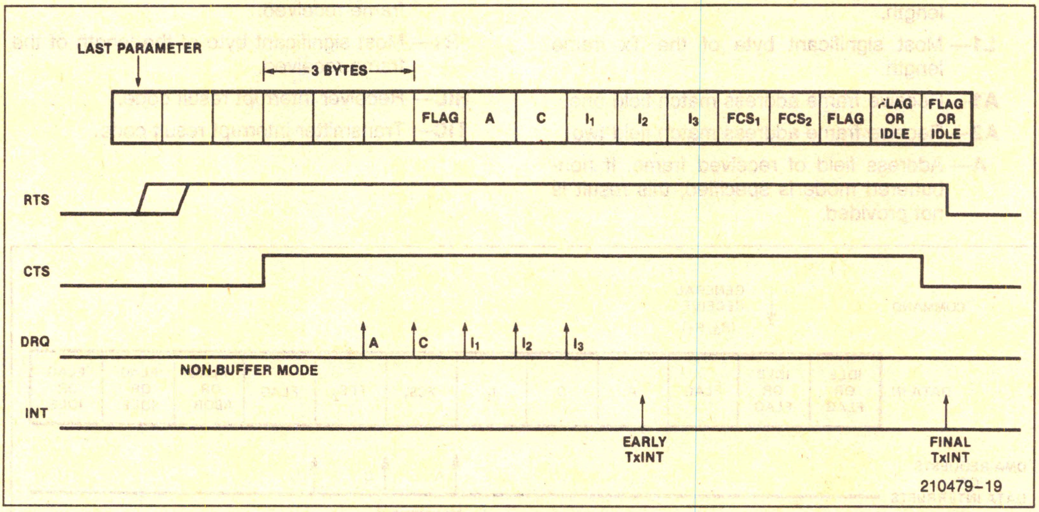


Figure 16b. Typical Frame Transmission, Non-Buffered Mode

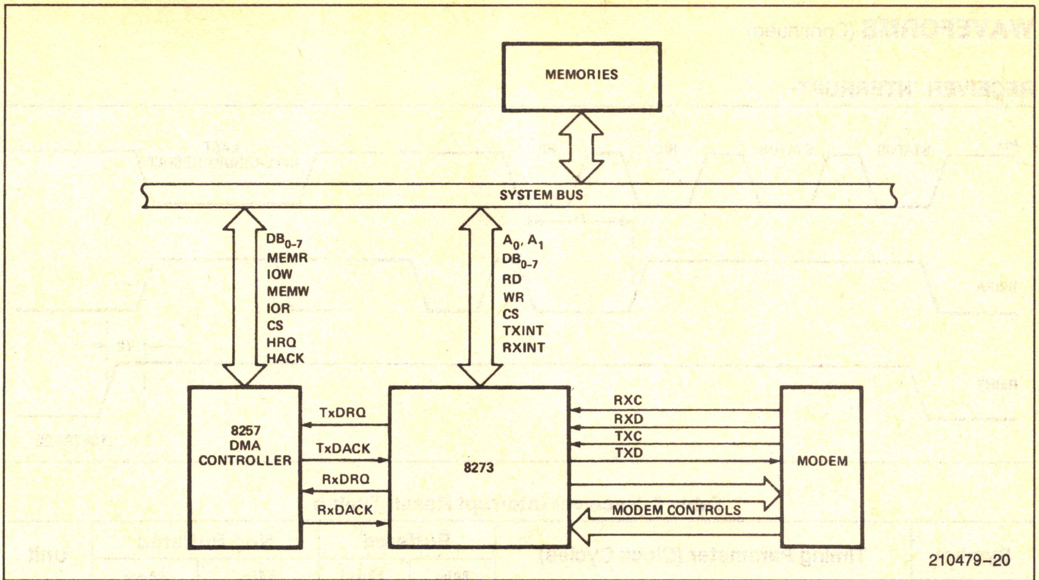


Figure 17. 8273 System Diagram

WAVEFORMS

COMMAND PHASE

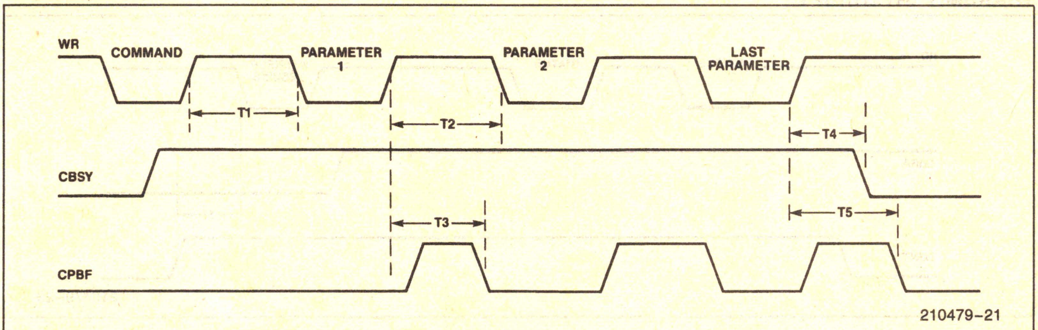


Table 2. Command Phase Timing (Full Duplex)

Symbol	Timing Parameter	Buffered		Non-Buffered		Unit
		Min	Max	Min	Max	
T1	Between Command & First Parameter	13	756	13	857	tcy
T2	Between Consecutive Parameters	10	604	10	705	tcy
T3	Command Parameter Buffer Full Bit Reset after Parmeter Loaded	10	604	10	705	tcy
T4	Command Busy Bit Reset after Last Parameter	128	702	128	803	tcy
T5	CPBF Bit Reset after Last Parameter	10	604	10	705	tcy

WAVEFORMS (Continued)

RECEIVER INTERRUPT

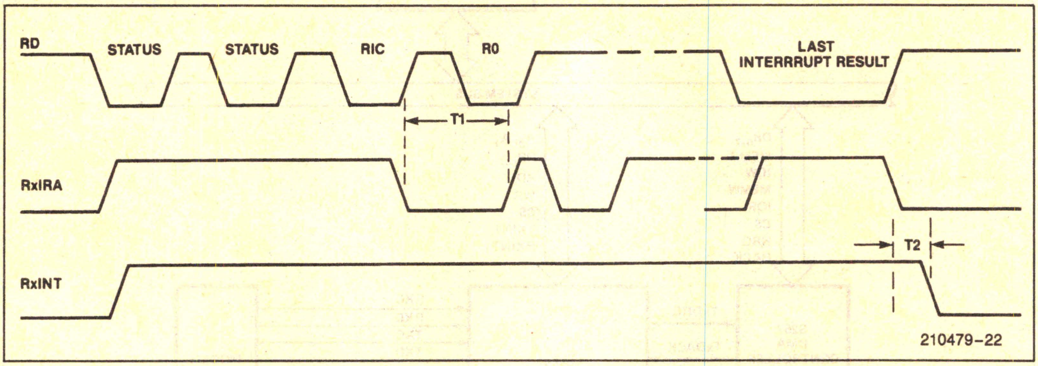


Table 3. Receiver Interrupt Result Timing

Symbol	Timing Parameter (Clock Cycles)	Buffered		Non-Buffered		Unit
		Min	Max	Min	Max	
T1	RxIRA Bit Set after RIC Read	18	29	18	29	tcy
T2	RxINT Goes Away after Last Int. Result Read	16	27	16	27	tcy

TRANSMIT INTERRUPT

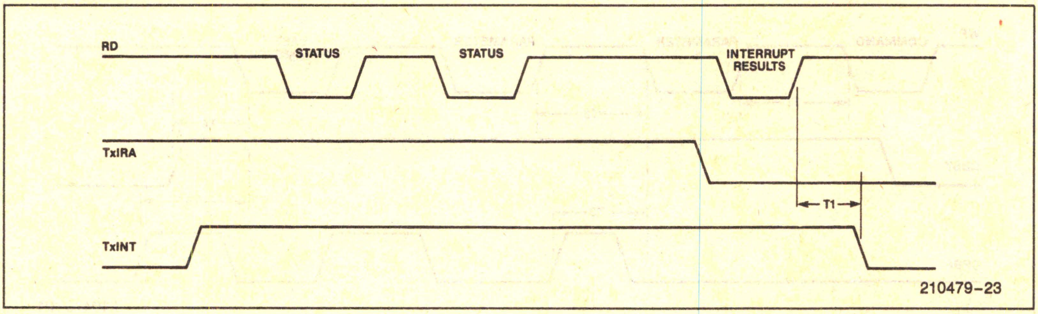


Table 4. Transmit Interrupt Result

Symbol	Timing (Clock Cycle)	Buffered		Non-Buffered		Unit
		Min	Max	Min	Max	
T1	TxINT Inactive after Int. Results Read	13	353	13	454	tcy

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin With Respect to Ground............. −0.5V to +7V
Power Dissipation .... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS 8273 (T_A = 0°C to 70°C, V_{CC} = +5.0V ±5%)

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA for Data Bus Pins I_{OL} = 1.0 mA for Output Port Pins I_{OL} = 1.6 mA for All Other Pins
V_{OH}	Output High Voltage	2.4		V	I_{OH} = − 200 μA for Data Bus Pins I_{OH} = −100 μA for All Other Pins
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		180	mA	

CAPACITANCE 8273 (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	t_c = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured Pins Returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ±5%)

CLOCK TIMING (8273)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{CY}	Clock	250		1000	ns	64K Baud Max Operating Rate
t_{CL}	Clock Low	120			ns	
t_{CH}	Clock High	120			ns	

A.C. CHARACTERISTICS 8273 (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 5%)

READ CYCLE

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{RD}$	0		ns	(Note 2)
t_{CA}	Select Hold from $\overline{RD}$	0		ns	(Note 2)
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	Data Delay from Address		300	ns	(Note 2)
t_{RD}	Data Delay from $\overline{RD}$		200	ns	C_L = 150 pF, (Note 2)
t_{DF}	Output Float Delay	20	100	ns	C_L = 20 pF For Minimum; 150 pF for Maximum
t_{DC}	DACK Setup to $\overline{RD}$	25		ns	
t_{CD}	DACK Hold from $\overline{RD}$	25		ns	
t_{KD}	Data Delay from DACK		300	ns	

WRITE CYCLE

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{WR}$	0		ns	
t_{CA}	Select Hold from $\overline{WR}$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$	150		ns	
t_{WD}	Data Hold from $\overline{WR}$	0		ns	
t_{DC}	DACK Setup to $\overline{WR}$	25		ns	
t_{CD}	DACK Hold from $\overline{WR}$	25		ns	

DMA

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CQ}	Request Hold from $\overline{WR}$ or $\overline{RD}$ (for Non-Burst Mode)		200	ns	

OTHER TIMING

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{RSTW}	Reset Pulse Width	10		t_{CY}	
t_r	Input Signal Rise Time		20	ns	
t_f	Input Signal Fall Time		20	ns	
t_{RSTS}	Reset to First $\overline{IOWR}$	2		t_{CY}	
t_{CY32}	32X Clock Cycle Time	$13.02 \times t_{CY}$		ns	
t_{CL32}	32X Clock Low Time	$4 \times t_{CY}$		ns	
t_{CH32}	32X Clock High Time	$4 \times t_{CY}$		ns	
t_{DPLL}	$\overline{DPLL}$ Output Low	$1 \times t_{CY} - 50$		ns	

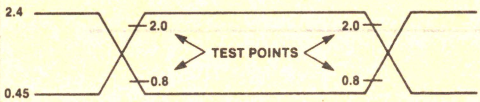
A.C. CHARACTERISTICS 8273 (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 5%) (Continued)

OTHER TIMING (Continued)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{DCL}	Data Clock Low	$1 \times t_{CY} - 50$		ns	
t_{DCH}	Data Clock High	$2 \times t_{CY}$		ns	
t_{DCY}	Data Clock	$62.5 \times t_{CY}$		ns	(Note 3)
t_{TD}	Transmit Data Delay		200	ns	
t_{DS}	Data Setup Time	200		ns	
t_{DH}	Data Hold Time	100		ns	
t_{FLD}	$\overline{\text{FLAG DET}}$ Output Low	$8 \times t_{CY} \pm 50$		ns	

NOTES:
1. All timing measurements are made at the reference voltages unless otherwise specified: Input "1" at 2.0V, "0" at 0.8V; Output "1" at 2.0V, "0" at 0.8V.
2. t_{AD}, t_{RD}, t_{AC}, and t_{CA} are not concurrent specs.
3. If receive commands or Read/Write Port commands are issued while both the transmitter and receiver are active, this specification will be 81.5 T_{CY} min.

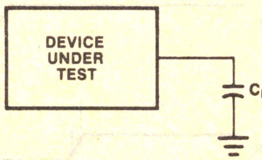
A.C. TESTING INPUT, OUTPUT WAVEFORM



210479–24

A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT

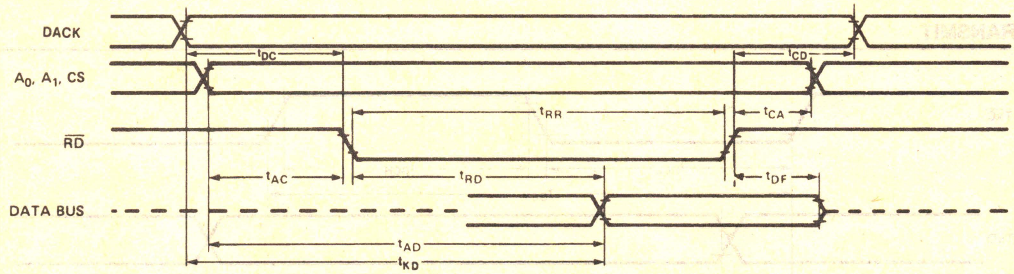


210479–25

C_L = 150 pF
C_L Includes Jig Capacitance

WAVEFORMS

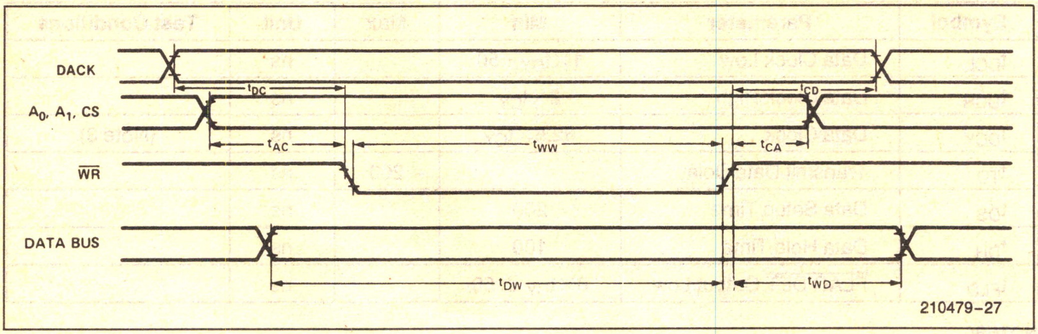
READ



210479–26

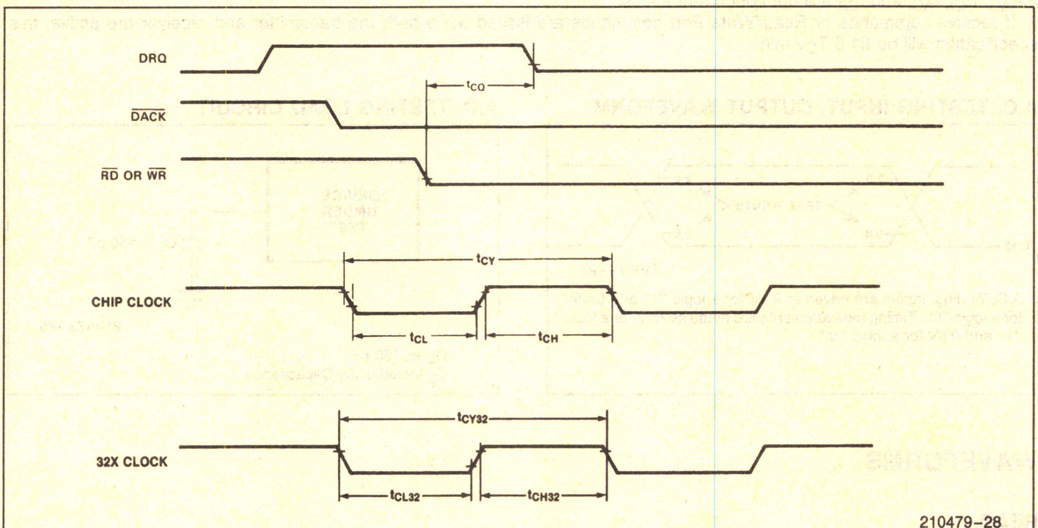
WAVEFORMS (Continued)

WRITE



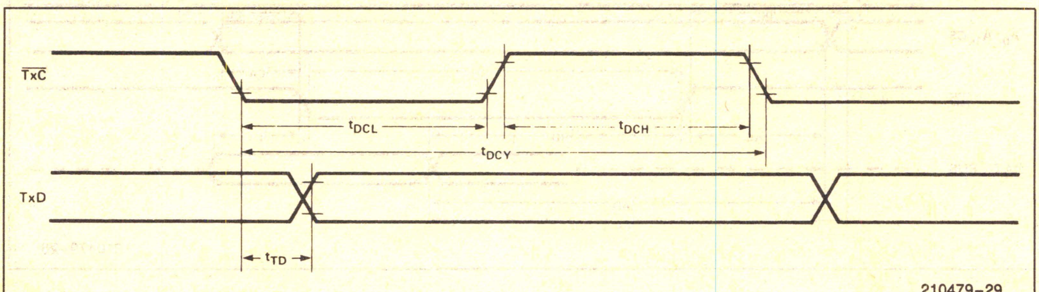
210479-27

DMA



210479-28

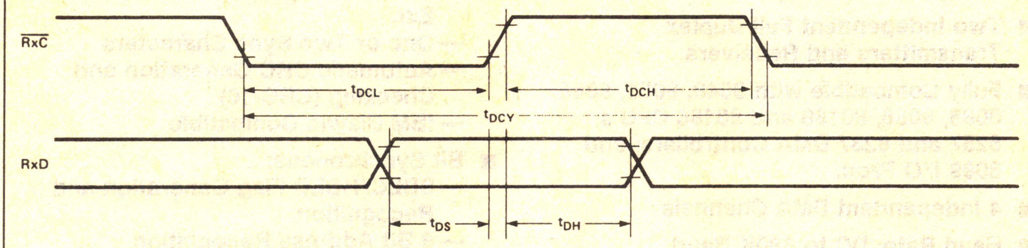
TRANSMIT



210479-29

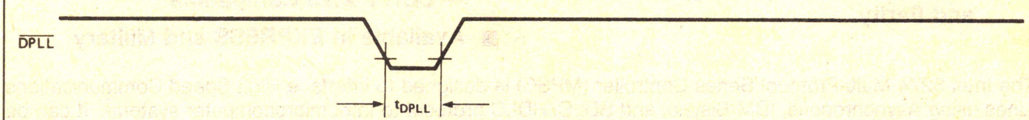
WAVEFORMS (Continued)

RECEIVE



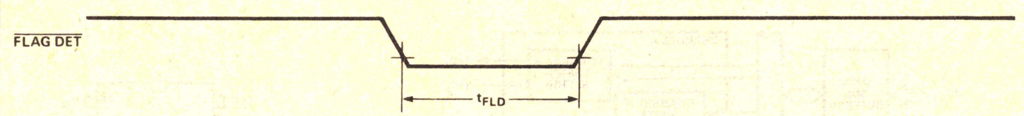
210479-30

DPLL OUTPUT



210479-31

FLAG DETECT OUTPUT



210479-1

8274
MULTI-PROTOCOL SERIAL CONTROLLER (MPSC)

- Asynchronous, Byte Synchronous and Bit Synchronous Operation
- Two Independent Full Duplex Transmitters and Receivers
- Fully Compatible with 8048, 8051, 8085, 8088, 8086, 80188 and 80186 CPU's; 8257 and 8237 DMA Controllers; and 8089 I/O Proc.
- 4 Independent DMA Channels
- Baud Rate: DC to 880K Baud
- Asynchronous:
  - 5–8 Bit Character; Odd, Even, or No Parity; 1, 1.5 or 2 Stop Bits
  - Error Detection: Framing, Overrun, and Parity
- Byte Synchronous:
  - Character Synchronization, Int. or Ext.
  - One or Two Sync Characters
  - Automatic CRC Generation and Checking (CRC-16)
  - IBM Bisync Compatible
- Bit Synchronous:
  - SDLC/HDLC Flag Generation and Recognition
  - 8 Bit Address Recognition
  - Automatic Zero Bit Insertion and Deletion
  - Automatic CRC Generation and Checking (CCITT-16)
  - CCITT X.25 Compatible
- Available in EXPRESS and Military

The Intel 8274 Multi-Protocol Series Controller (MPSC) is designed to interface High Speed Communications Lines using Asynchronous, IBM Bisync, and SDLC/HDLC protocol to Intel microcomputer systems. It can be interfaced with Intel's MCS-48, -85, -51; iAPX-86, -88, -186 and -188 families, the 8237 DMA Controller, or the 8089 I/O Processor in polled, interrupt driven, or DMA driven modes of operation.

The MPSC is a 40 pin device fabricated using Intel's High Performance HMOS Technology.

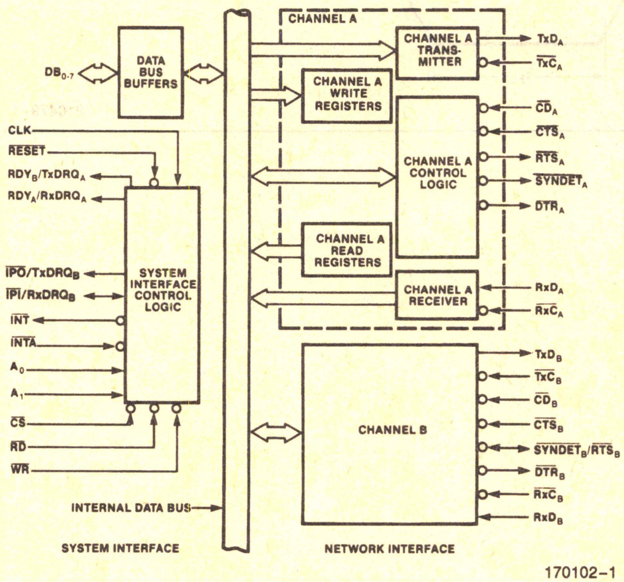
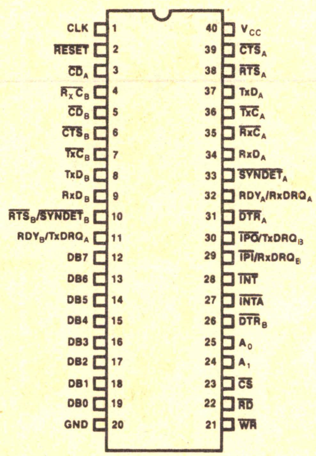


Figure 1. Block Diagram



Pin	Name	Pin	Name
1	CLK	40	V_{CC}
2	RESET	39	$\overline{CTS}_A$
3	$\overline{CD}_A$	38	$\overline{RTS}_A$
4	$\overline{R_xC}_B$	37	TxD_A
5	$\overline{CD}_B$	36	$\overline{TxC}_A$
6	$\overline{CTS}_B$	35	$\overline{RxC}_A$
7	$\overline{TxC}_B$	34	RxD_A
8	TxD_B	33	$\overline{SYNDET}_A$
9	RxD_B	32	$RDY_A/RxDRQ_A$
10	$\overline{RTS}_B/\overline{SYNDET}_B$	31	$\overline{DTR}_A$
11	$RDY_B/TxDRQ_A$	30	$\overline{IPO}/TxDRQ_B$
12	DB7	29	$\overline{IPI}/RxDRQ_B$
13	DB6	28	$\overline{INT}$
14	DB5	27	$\overline{INTA}$
15	DB4	26	$\overline{DTR}_B$
16	DB3	25	A_0
17	DB2	24	A_1
18	DB1	23	$\overline{CS}$
19	DB0	22	$\overline{RD}$
20	GND	21	$\overline{WR}$

Figure 2. Pin Configuration

November 1986
Order Number: 170102-002

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
CLK	1	I	**CLOCK:** System clock, TTL compatible.
$\overline{RESET}$	2	I	**RESET:** A low signal on this pin will force the MPSC to an idle state. TxD_A and TxD_B are forced high. The modem interface output signals are forced high. The MPSC will remain idle until the control registers are initialized. Reset must be true for one complete CLK cycle.
$\overline{CD}_A$	3	I	**CARRIER DETECT (CHANNEL A):** This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD_A line. If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD}_A$ has been activated.
$\overline{RxC}_B$	4	I	**RECEIVE CLOCK (CHANNEL B):** The serial data are shifted into the Receive Data input (RxD_B) on the rising edge of the Receive Clock.
$\overline{CD}_B$	5	I	**CARRIER DETECT (CHANNEL B):** This interface signal is supplied by the modem to indicated that a data carrier signal has been detected and that a valid data signal is present on the RxD_B line. If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD}_B$ has been activated.
$\overline{CTS}_B$	6	I	**CLEAR TO SEND (CHANNEL B):** This interface signal is supplied by the modem in response to an active $\overline{RTS}$ signal. $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated.
$\overline{TxC}_B$	7	I	**TRANSMIT CLOCK (CHANNEL B):** The serial data are shifted out from the Transmit Data output (TxD_B) on the falling edge of the Transmit Clock.
TxD_B	8	O	**TRANSMIT DATA (CHANNEL B):** This pin transmits serial data to the communications channel (Channel B).
RxD_B	9	I	**RECEIVE DATA (CHANNEL B):** This pin receives serial data from the communications channel (Channel B).
$\overline{SYNDET}_B$ $/\overline{RTS}_B$	10	I/O	**SYNCHRONOUS DETECTION (CHANNEL B):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating Flag detection. In asynchronous mode it is a general purpose input (Channel B). **REQUEST TO SEND (CHANNEL B):** General purpose output, generally used to signal that Channel B is ready to send data. When the RTS bit is reset in asynchronous mode, the signal does not go inactive (High) until the transmitter is empty. $\overline{SYNDET}_B$ or $\overline{RTS}_B$ selection is done by WR2; D7. (Channel A).

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RDY_B/ $TxDRQ_A$	11	O	**READY (CHANNEL B)/TRANSMITTER DMA REQUEST (CHANNEL A):** In mode 0 this pin is RDY_B and is used to synchronize data transfers between the processor and the MPSC (Channel B). In modes 1 and 2 this pin is $TxDRQ_A$ and is used by the Channel A transmitter to request a DMA transfer.
DB7	12	I/O	**DATA BUS:** The Data Bus lines are bidirectional three state lines which interface with the system's Data Bus.
DB6	13		
DB5	14		
DB4	15		
DB3	16		
DB2	17		
DB1	18		
DB0	19		
GND	20		**GROUND.**
V_{CC}	40		**POWER:** +5V Supply
$\overline{CTS_A}$	39	I	**CLEAR TO SEND (CHANNEL A):** This interface signal is supplied by the Modem in response to an active $\overline{RTS}$ signal. $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data. In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated.
$\overline{RTS_A}$	38	O	**REQUEST TO SEND (CHANNEL A):** General purpose output commonly used to signal that Channel A is ready to send data. When the RTS bit is reset in asynchronous mode, the signal does not go inactive (High) until the transmitter is empty.
TxD_A	37	O	**TRANSMIT DATA (CHANNEL A):** This pin transmits serial data to the commmunications channel (Channel A).
$\overline{TxC_A}$	36	I	**TRANSMIT CLOCK (CHANNEL A):** The serial data are shifted out from the Transmit Data output (TxD_A) on the falling edge of the Transmit Clock.
$\overline{RxC_A}$	35	I	**RECEIVE CLOCK (CHANNEL A):** The serial data are shifted into the Receive Data input (RxD_A) on the rising edge of the Receive Clock.
RxD_A	34	I	**RECEIVE DATA (CHANNEL A):** This pin receives serial data from the communications channel (Channel A).
$\overline{SYNDET_A}$	33	I/O	**SYNCHRONOUS DETECTION (CHANNEL A):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input). In SDLC mode, this pin is an output indicating flag detection. In asynchronous mode it is a general purpose input (Channel A).

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RDY_A/ $RxDRQ_A$	32	O	**READY:** In mode 0 this pin is RDY_A and is used to synchronize data transfers between the processor and the MPSC (Channel A). In modes 1 and 2 this pin is $RxDRQ_A$ and is used by the channel A receiver to request a DMA transfer.
$\overline{DTR_A}$	31	O	**DATA TERMINAL READY (CHANNEL A):** General purpose output.
$\overline{IPO}$/ $TxDRQ_B$	30	O	**INTERRUPT PRIORITY OUT/TRANSMITTER DMA REQUEST (CHANNEL B):** In modes 0 and 1, this pin is Interrupt Priority Out. It is used to establish a hardware interrupt priority scheme with $\overline{IPI}$. It is low only if $\overline{IPI}$ is low and the controlling processor is not servicing an interrupt from this MPSC. In mode 2 it is $TxDRQ_B$ and is used to request a DMA cycle for a transmit operation (Channel B).
$\overline{IPI}$/ $RxDRQ_B$	29	I/O	**INTERRUPT PRIORITY IN/RECEIVER DMA REQUEST (CHANNEL B):** In modes 0 and 1, $\overline{IPI}$ is Interrupt Priority In. A low on $\overline{IPI}$ means that no higher priority device is being serviced by the controlling processor's interrupt service routine. In mode 2 this pin is $RxDRQ_B$ and is used to request a DMA cycle for a receive operation (Channel B).
$\overline{INT}$	28	O	**INTERRUPT:** The interrupt signal indicates that the highest priority internal interrupt requires service (open collector). Priority can be resolved via an external interrupt controller or a daisy-chain scheme.
$\overline{INTA}$	27	I	**INTERRUPT ACKNOWLEDGE:** This Interrupt Acknowledge signal allows the highest priority interrupting device to generate an interrupt vector. This pin must be pulled high (inactive) in non-vector mode.
$\overline{DTR_B}$	26	O	**DATA TERMINAL READY (CHANNEL B):** This is a general purpose output.
A_0	25	I	**ADDRESS:** This line selects Channel A or B during data or command transfers. A low selects Channel A.
A_1	24	I	**ADDRESS:** This line selects between data or command information transfer. A low means data.
$\overline{CS}$	23	I	**CHIP SELECT:** This signal selects the MSPC and enables reading from or writing into registers.
$\overline{RD}$	22	I	**READ:** Read controls a data byte or status byte transfer from the MPSC to the CPU.
$\overline{WR}$	21	I	**WRITE:** Write controls transfer of data or commands to the MPSC.

RESET

When the 8274 $\overline{\text{RESET}}$ line is activitated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ($\overline{\text{RTS}}$, $\overline{\text{DTR}}$) are forced high. In addition, the pointers registers are set to zero.

GENERAL DESCRIPTION

The Intel 8274 Multi-Protocol Serial Controller is a microcomputer peripheral device which supports Asynchronous, Byte Synchronous (Monosync, IBM Bisync), and Bit Synchronous (ISO's HDLC, IBM's SDLC) protocols. This controller's flexible architecture allows easy implementation of many variations of these three protocols with low software and hardware overhead.

The Multi-Protocol Serial controller (MPSC) implements two independent serial receiver/transmitter channels.

The MPSC supports several microprocessor interface options: Polled, Wait, Interrupt driven and DMA driven. The MPSC is designed to support INTEL's MCS-85 and iAPX 86, 88, 186, 188 families.

FUNCTIONAL DESCRIPTION

Additional information on Asynchronous and Synchronous Communications with the 8274 is available respectively in the Applications Notes AP 134 and AP 145.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B).

In the following discussion, the writable registers will be referred to as WRO through WR7 and the readable registers will be referred to as RRO through RR2.

This section of the data sheet describes how the Asynchronous and Synchronous protocols are implemented in the MPSC. It describes general considerations, transmit operation, and receive operation for Asynchronous, Byte Synchronous, and Bit Synchronous protocols.

ASYNCHRONOUS OPERATIONS

Transmitter/Receiver Initialization

(See Detailed Command Description Section for complete information)

In order to operate in asynchronous mode, each MPSC channel must be initialized with the following information:

1. Transmit/Receive Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 1, 16, 32, or 64 times the data-link bit rate. If the X1 clock mode is selected, the bit synchronization must be accomplished externally.
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2.
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4.
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3.
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3.
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 0).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data.

Byte Written								Number of Bits Transmitted
D7	D6	D5	D4	D3	D2	D1	D0	(Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5.
8. Interrupt Mode. Specified by bits 3 and 4 of WR1.

For data transmission via a modem or RS-232-C interface, the following information must also be specified:

1. The Request To Send (RTS) (WR5; D1) and Data Terminal Ready (DTR) (WR5; D7) bits must be set along with the Transmit Enable bit (WR5; D3).
2. Auto Enable may be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. However, the Transmit Enable bit (WR3; D3) and Receive Enable bit (WR3; D1) must be set in order to use the Auto Enable mode. Auto Enable is controlled by bit 5 of WR3.

When loading Initialization parameters into the MPSC, WR4 information must be written before the WR1, WR3, WR5 parameters commands.

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Intel Application Note AP 134.

TRANSMIT

The transmit function begins when the Transmit Enable bit (WR5; D3) is set. The MPSC automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits (1, 1.5 or 2 bits) to the data character being transmitted. 1.5 stop bits option must be used with X16, X32 or X64 clock options only. The data character is transmitted least significant bit first.

The serial data are shifted out from the Transmit Data (TxD) output on the falling edge of the Transmit Clock (TxC) input at a rate programmable to 1, $1/16$th, $1/32$nd, or $1/64$th of the clock rate supplied to the TxC input.

The TxD output is held high when the transmitter has no data to send, unless, under program control, the Send Break (WR5; D4) command is issued to hold the TxD low.

If the External/Status Interrupt bit (WR1; D0) is set, the status of $\overline{CD}$, $\overline{CTS}$ and $\overline{SYNDET}$ are monitored and, if any changes occur for a period of time greater than the minimum specified pulse width, an interrupt is generated. $\overline{CTS}$ is usually monitored using this interrupt feature (e.g., Auto Enable option).

The Transmit Buffer Empty bit (RR0; D2) is set by the MPSC when the data byte from the buffer is loaded in the transmit shift register. Data should be written to the MPSC only when the Tx buffer becomes empty to prevent overwriting.

Receive

The receive function begins when the Receive Enable (WR3; D0) bit is set. If the Auto Enable (WR3; D5) option is selected, then Carrier Detect ($\overline{CD}$) must also be low. A valid start bit is detected if a low persists for at least ½ bit time on the Receive Data (RxD) input.

The data is sampled at mid-bit time, on the rising edge of RxC, until the entire character is assembled. The receiver inserts 1's when a character is less than 8 bits. If parity (WR4; D0) is enabled and the character is less than 8 bits the parity bit is not stripped from the character.

Error Reporting

The receiver also stores error status for each of the 3 data characters in the data buffer. Three error conditions may be encountered during data reception in the asynchronous mode:

1. **Parity.** If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4). When a parity error is detected, the parity error flag (RR1; D4) is set and remains set until it is reset by the Error Reset command (WR0; D5, D4, D3).
2. **Framing.** A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled). When a Framing Error is detected, the Framing Error bit (RR1; D6) is set and remains set until reset by the Error Reset Command (WR0; D5, D4, D3). The detection of a Framing Error adds an additional ½ bit time to the character time so the Framing Error is not interpreted as a new start bit.
3. **Overrun.** If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1; D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. Only the overwritten character is flagged with the Receive Overrun bit. The Receive Overrun bit (RR1; D5) is reset by the Error Reset command (WR0; D5, D4, D3).

External/Status Latches

The MPSC continuously monitors the state of five external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).
5. TxUNDERRUN/EOM—Transmitter Underrun/ End of Message.

   A change of state in any of these monitored conditions will cause the associated status bit in RR0 to be latched (and optionally cause an interrupt).

   If the External/Status Interrupt bit (WR1; D0) is enabled, Break Detect (RR0; D7) and Carrier Detect (RR0; D3) will cause an interrupt. Reset External/Status interrupts (WR0; D5, D4, D3) will clear Break Detect and Carrier Detect bits if they are set.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B). They are all accessed via the command ports.

An internal pointer register selects which of the command or status registers will be read or written during a commmand/status access of an MPSC channel.

After reset, the contents of the pointer registers are zero. The first write to a command register causes the data to be loaded into Write Register 0 (WR0). The three least significant bits of WR0 are loaded into the Command/Status Pointer. The next read or write operation accesses the read or write register selected by the pointer. The pointer is reset after the read or write operation is completed.

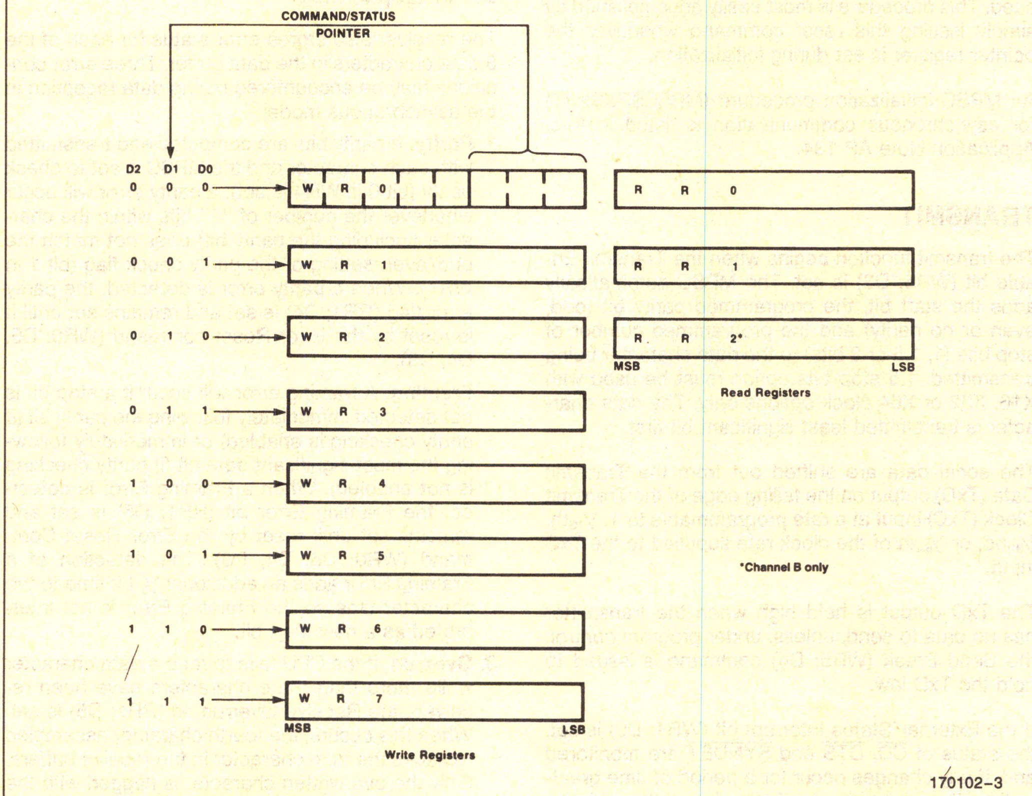


Figure 3. Command/Status Register Architecture (each serial channel)

Asynchronous Mode Register Setup

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Tx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	0	0	0	0	Rx ENABLE
WR4	00 X1 Clock 01 X16 Clock 10 X32 Clock 11 X64 Clock		0	0	01 1 STOP BIT 10 1½ STOP BITS 11 2 STOP BITS		EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0	RTS	0

SYNCHRONOUS OPERATION—MONOSYNC, BISYNC

General

The MPSC must be initialized with the following parameters: odd or even parity (WR4; D1, D0), X1 clock mode (WR4; D7, D6), 8- or 16-bit sync character (WR4; D5, D4), CRC polynomial (WR5; D2), Transmitter Enable (WR5; D3), interrupt modes (WR1, WR2), transmit character length (WR5; D6, D5) and receive character length (WR3; D7, D6). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The data is transmitted on the falling edge of the Transmit Clock, ($\overline{\text{TxC}}$) and is received on the rising edge of Receive Clock ($\overline{\text{RxC}}$). The X1 clock is used for both transmit and receive operations for all three sync modes: Mono, Bi and External.

Transmit Set-Up—Monosync, Bisync

Transmit data is held high after channel reset, or if the transmitter is not enabled. A break may be programmed to generate a spacing line that begins as soon as the Send Break (WR5; D4) bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

Using interrupts for data transfer requires that the Transmit Interrupt/DMA Enable bit (WR1; D1) be set. An interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied either by writing another character into the transmitter or by resetting the Transmitter Interrupt/DMA Pending latch with a Reset Transmitter Inter-

Synchronous Mode Register Setup—Monosync, Bisync

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Tx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	Rx ENABLE
WR4	0	0	00 8 bit Sync 01 16 bit Sync 11 Ext Sync		0	0	EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	1 (SELECTS CRC-16)	RTS	Tx CRC ENABLE

rupt/DMA Pending Command (WR0; D5, D4, D3). If nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupt, but this situation does cause a Transmit Underrun condition (RR0; D6).

Data Transfers using the RDY signal are for software controlled data transfers such as block moves. RDY tells the CPU that the MPSC is not ready to accept/provide data and that the CPU must extend the output/input cycle. DMA data transfers use the TxDRQ A/B signals which indicate that the transmit buffer is empty, and that the MPSC is ready to accept the next data character. If the data character is not loaded into the MPSC by the time the transmit shift register is empty, the MPSC enters the Transmit Underrun condition.

The MPSC has two programmable options for solving the transmit underrun condition: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0; D6) is in a set condition allowing the insertion of sync characters when there is no data to send. The CRC is not calculated on these automatically inserted sync characters. When the CPU detects the end message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data to send.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded in the Transmit Shift Register. The status register indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded into the Tx Shift Register). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5; D3).

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5; D0) indicates CRC accumulation when the program sends the first data character to the MPSC. Although the MPSC automatically transmits up to two sync characters (16 bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The Transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded into the transmit shift register. To ensure this bit in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the MPSC.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16 bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the MPSC.

In the transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16) (WR4; D5, D4). When in the Monosync mode, the transmitter sends from WR6 and the receiver compares against WR7. One or two CRC polynomials, CRC 16 or SDLC, may be used with synchronous modes. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command (WR0; D7, D6).

The External/Status interrupt (WR1; D0) mode can be used to monitor the status of the $\overline{CTS}$ input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enable (WR3; D5) feature can be used to enable the transmitter when $\overline{CTS}$ is active. The first data transfer to the MPSC can begin when the External/Status interrupt (CTS (RR0; D5) status bit set) occurs following the Transmit Enable command (WR5; D3).

Receive

After a channel reset, the receiver is in the Hunt phase, during which the MPSC looks for character synchronization. The Hunt begins only when the receiver is enabled and data transfer begins only when character synchronization has been achieved. If character synchronization is lost, the hunt phase can be re-entered by writing the Enter Hunt Phase (WR3; D4) bit. The assembly of received data continues until the MPSC is reset or until the receiver is disabled (by command or by $\overline{CD}$ while in the Auto Enables mode) or until the CPU sets the Enter Hunt Phase bit. Under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit (WR3; D1) bit. After character synchronization is achieved the assembled characters are transferred to the receive data FIFO. After

receiving the first data character, the Sync Character Load Inhibit bit should be reset to zero so that all characters are received, including the sync characters. This is important because the received CRC may look like a sync character and not get received.

Data may be transferred with or without interrupts. Transferring data without interrupts is used for a purely polled operation or for off-line conditions. There are two interrupt modes available for data transfer: Interrupt on First Character Only and Interrupt on Every Character.

Interrupt on First Character Only mode is normally used to start a polling loop, a block transfer sequence using RDY to synchronize the CPU to the incoming data rate, or a DMA transfer using the RxDRQ signal. The MPSC interrupts on the first character and thereafter only interrupts after a Special Receive Condition is detected. This mode can be reinitialized using the Enable Interrupt On Next Receive Character (WR0; D5, D4, D3) command which allows the next character received to generate an interrupt. Parity Errors do not cause interrupts, but End of Frame (SDLC operation) and Receive Overrun do cause interrupts in this mode. If the external status interrupts (WR1; D0) are enabled an interrupt may be generated any time the $\overline{CD}$ changes state.

Interrupt On Every Character mode generates an interrupt whenever a character enters the receive buffer. Errors and Special Receive Conditions generate a special vector if the Status Affects Vector (WR1B; D2) is selected. Also the Parity Error may be programmed (WR1; D4, D3) not to generate the special vector while in the Interrupt On Every Character mode.

The Special Receive Condition interrupt can only occur while in the Receive Interrupt On First Character Only or the Interrupt On Every Receive Character modes. The Special Receive Condition interrupt is caused by the Receive Overrun (RR1; D5) error condition. The error status reflects an error in the current word in the receive buffer, in addition to any Parity or Overrun errors since the last Error Reset (WR0; D5, D4, D3). The Receive Overrun and Parity error status bits are latched and can only be reset by the Error Reset (WR0; D5, D4, D3) command.

The CRC check result may be obtained by checking for CRC bit (RR1; D6). This bit gives the valid CRC result 16 bit times after the second CRC byte has been read from the MPSC. After reading the second CRC byte, the user software must read two more characters (may be sync characters) before checking for CRC result in RR1. Also for proper CRC computation by the receiver, the user software must reset the Receive CRC Checker (WR0; D7, D6) after receiving the first valid data character. The receive CRC Enable bit (WR3; D3) may also be enabled at this time.

SYNCHRONOUS OPERATION—SDLC

General

Like the other synchronous operations the SDLC mode must be initialized with the following parameters: SDLC mode (WR4; D5, D4), SDLC polynomial (WR5; D2), Request to Send, Data Terminal Ready, transmit character length (WR5; D6, D5), interrupt modes (WR1; WR2), Transmit Enable (WR5; D3), Receive Enable (WR3; D0), Auto Enable (WR3; D5) and External/Status Interrupt (WR1; D0). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The Interrupt modes for SDLC operation are similar to those discussed previously in the synchronous operations section.

Synchronous Mode Register Setup—SDLC/HDLC

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx
WR4	0	0	1	0	0	0	0	0
			(SELECTS SDLC/ HDLC MODE)					
WR5	DTR	00 Tx ≤ 5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0 (SELECTS SDLC/HDLC CRC)	RTS	Tx CRC ENABLE

Transmit

After a channel reset, the MPSC begins sending SDLC flags.

Following the flags in an SDLC operation the 8-bit address field, control field and information field may be sent to the MPSC by the microprocessor. The MPSC transmits the Frame Check Sequence using the Transmit Underrun feature. The MPSC automatically inserts a zero after every sequence of 5 consecutive 1's except when transmitting Flags or Aborts.

SDLC—like protocols do not have provision for fill characters within a message. The MPSC therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by sending the two bytes of CRC and then one or more flags. This allows very high-speed transmissions under DMA or CPU control without requiring the CPU to respond quickly to the end-of-message situation.

After a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Flag characters are sent. The MPSC begins to send the frame when data is written into the transmit buffer. Between the time the first data byte is written, and the end of the message, the Reset Transmit Underrun/EOM (WR0; D7, D6) command must be issued. The Transmit Underrun/EOM status bit (RR0; D6) is in the reset state at the end of the message which automatically sends the CRC characters.

The MPSC may be programmed to issue a Send Abort command (WR0; D5, D4, D3). This command causes at least eight 1's but less than fourteen 1's to be sent before the line reverts to continuous flags.

Receive

After initialization, the MPSC enters the Hunt phase, and remains in the Hunt phase until the first Flag is received. The MPSC never again enters the Hunt phase unless the microprocessor writes the Enter Hunt command. The MPSC will also detect flags separated by a single zero. For example, the bit pattern 011111101111110 will be detected as two flags.

The MPSC can be programmed to receive all frames or it can be programmed to the Address Search Mode. In the Address Search Mode, only frames with addresses that match the value in WR6 or the global address (0FFH) are received by the MPSC. Extended address recognition must be done by the microprocessor software.

The control and information fields are received as data.

SDLC/HDLC CRC calculation does not have an 8-bit delay, since all characters are included in the calculation, unlike Byte Synchronous Protocols.

Reception of an abort sequence (7 or more 1's) will cause the Break/Abort bit (RR0; D7) to be set and will cause an External/Status interrupt, if enabled. After the Reset External/Status Interrupts Command has been issued, a second interrupt will occur at the end of the abort sequence.

MPSC

Detailed Command/Status Description

GENERAL

The MPSC supports an extremely flexible set of serial and system interface modes.

The system interface to the CPU consists of 8 ports or buffers:

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedance	High Impedance

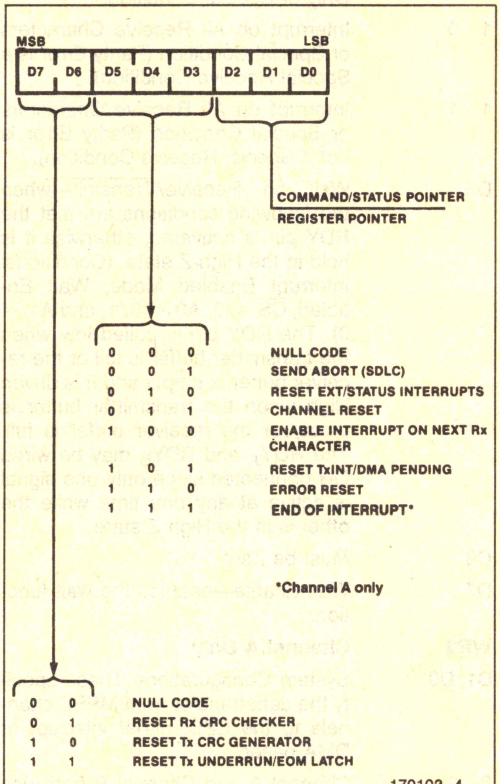
Data buffers are addressed by $A_1 = 0$, and Command ports are addressed by $A_1 = 1$.

COMMAND/STATUS DESCRIPTION

The following command and status bytes are used during initialization and execution phases of operation. All Command/Status operations on the two channels are identical, and independent, except where noted.

Detailed Register Description

Write Register 0 (WR0):



170102-4

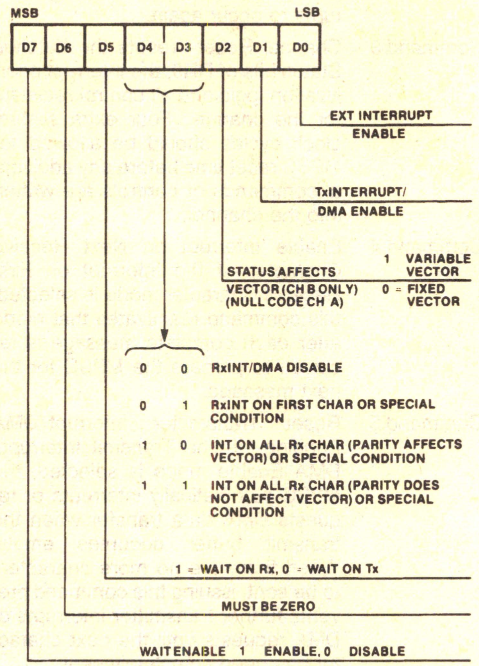
WR0

D2, D1, D0—Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5, D4, D3—Command bits determine which of the basic seven commands are to be performed.

Command 0	Null—has no effect.
Command 1	Send Abort—causes the generation of eight to thirteen 1's when in the SDLC mode.
Command 2	Reset External/Status Interrupts—resets the latched status bits of RR0 and re-enables them, allowing interrupts to occur again.
Command 3	Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.
Command 4	Enable Interrupt on Next Receive Character—if the Interrupt on First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.
Command 5	Reset Transmitter Interrupt/DMA Pending—if The Transmit Interrupt/DMA Enable mode is selected, the MPSC automatically interrupts or requests DMA data transfer when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts or DMA requests until the next character has been completely sent.
Command 6	Error Reset—error latches, Parity and Overrun errors in RR1 are reset.
Command 7	End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.
D7, D6	CRC Reset Code.
00	Null—has no effect.
01	Reset Receive CRC Checker—resets the CRC checker to 0's. If in SDLC mode the CRC checker is initialized to all 1's.
10	Reset Transmit CRC Generator—resets the CRC generator to 0's. If in SDLC mode the CRC generator's initialized to all 1's.
11	Reset Tx Underrun/End of Message Latch.

Write Register 1 (WR1):



170102-5

WR1

D0 — External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 — Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 — Status Affects vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit if set then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4, D3 — Receive Interrupt Mode.

0 0 — Receive Interrupts/DMA Disabled.

0 1 — Receive Interrupt on First Character Only or Special Condition.

1 0 — Interrupt on All Receive Characters or Special Condition (Parity Error is a Special Receive Condition).

1 1 — Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 — Wait on Receive/$\overline{\text{Transmit}}$—when the following conditions are met the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}$ = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired OR connected since only one signal is active at any one time while the other is in the High Z state.

D6 — Must be Zero.

D7 — Wait Enable—enables the wait function.

WR2 — **Channel A Only**

D1, D0 — System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 — Channel A and Channel B both use interrupts.

0 1 — Channel A uses DMA, Channel B uses interrupts.

1 0 — Channel A and Channel B both use DMA.

1 1 — Illegal Code.

D2 — Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

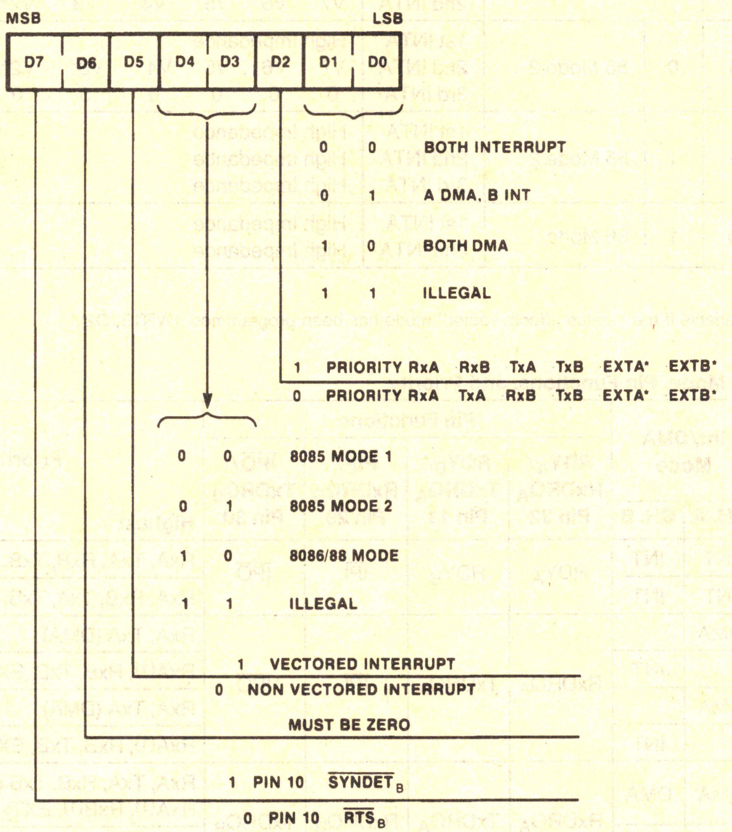
0 — (Highest) RxA, TxA, RxB, TxB, ExTA, ExTB (Lowest).

1 — (Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).

D5, D4, D3 — Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table.)

0 X X	Non-vectored interrupts—intended for use with external DMA CONTROLLER. The Data Bus remains in a high impedance state during INTA sequences.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy chained priority structure. (See System Interface section).
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy chained priority structure. (See System Interface section).
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy chained priority structure. (See System Interface section).
D6	Must be zero.
D7	zero Pin 10 = $\overline{RTS_B}$ one Pin 10 = $\overline{SYNDET_B}$

Write Register 2 (WR2): Channel A Only



170102-6

NOTE:
*External Status Interrupt only if EXT Interrupt Enable (WR1; D0) is set.

The following table describes the MPSC's response to an interrupt acknowledge sequence:

D5	D4	D3	$\overline{IPI}$	MODE	INTA	Data Bus							
0	X	X	X	Non-vectored	Any INTA	D7 High Impedance							D0
1	0	0	0	85 Mode 1	1st INTA	1	1	0	0	1	1	0	1
					2nd INTA	V7	V6	V5	V4*	V3*	V2*	V1	V0
					3rd INTA	0	0	0	0	0	0	0	0
1	0	0	1	85 Mode 1	1st INTA	1	1	0	0	1	1	0	1
					2nd INTA	High Impedance							
					3rd INTA	High Impedance							
1	1	0	0	86 Mode	1st INTA	High Impedance							
					2nd INTA	V7	V6	V5	V4	V3	V2*	V1*	V0*
1	0	1	0	85 Mode 2	1st INTA	High Impedance							
					2nd INTA	V7	V6	V5	V4*	V3*	V2*	V1	V0
					3rd INTA	0	0	0	0	0	0	0	0
1	0	1	1	85 Mode 2	1st INTA	High Impedance							
					2nd INTA	High Impedance							
					3rd INTA	High Impedance							
1	1	0	1	86 Mode	1st INTA	High Impedance							
					2nd INTA	High Impedance							

NOTE:
*These bits are variable if the "status affects vector" mode has been programmed, (WR1B, D2).

Interrupt/DMA Mode, Pin Functions, and Priority

Ch. A WR2			Int/DMA Mode		Pin Functions				Priority
D_2	D_1	D_0	CH. A	CH. B	RDY_A/ $RxDRQ_A$ Pin 32	RDY_B/ $TxDRQ_A$ Pin 11	$\overline{PIP}$/ $RxDRQ_B$ Pin 29	$\overline{IPO}$/ $TxDRQ_B$ Pin 30	Highest — Lowest
0	0	0	INT	INT	RDY_A	RDY_B	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA, RxB, TxB, EXT_A, EXT_B
1	0	0	INT	INT					RxA, RxB, TxA, TxB, EXT_A, EXT_B
0	0	1	DMA		$RxDRQ_A$	$TxDRQ_A$	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA (DMA)
				INT					RxA[1], RxB, TxB, EXT_A, EXT_B (INT)
1	0	1	DMA						RxA, TxA (DMA)
				INT					RxA[1], RxB, TxB, EXT_A, EXT_B (INT)
0	1	0	DMA	DMA	$RxDRQ_A$	$TxDRQ_A$	$RxDRQ_B$	$TxDRQ_B$	RxA, TxA, RxB, TxB (DMA) RxA[1], RxB[1], EXT_A, EXT_B, (INT)
1	1	0	DMA	DMA					RxA, RxB, TxA, TxB, (DMA) RxA[1], RxB[1], EXT_A, EXT_B (INT)

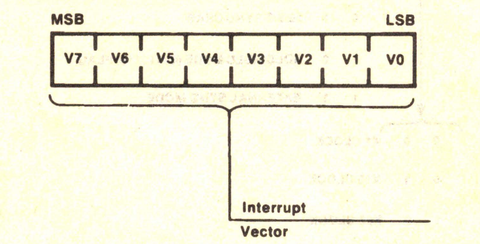
NOTE:
1. Special Receive Condition

Interrupt Vector Mode Table

8085 Modes 8086/88 Mode	V_4 V_2	V_3 V_1	V_2 V_0	Channel	Condition
	0	0	0	B	Tx Buffer Empty
	0	0	1		Ext/Status Change
	0	1	0		Rx Char. Available
	0	1	1		Special Rx Condition (Note 1)
	1	0	0	A	Tx Buffer Empty
	1	0	1		Ext/Status Change
	1	1	0		Rx Char. Available
	1	1	1		Special Rx Condition (Note 1)

NOTE:
1. Special Receive Condition = Parity Error, Rx Overrun Error, Framing Error, End of Frame (SDLC).

Write Register 2 (WR2): Channel B

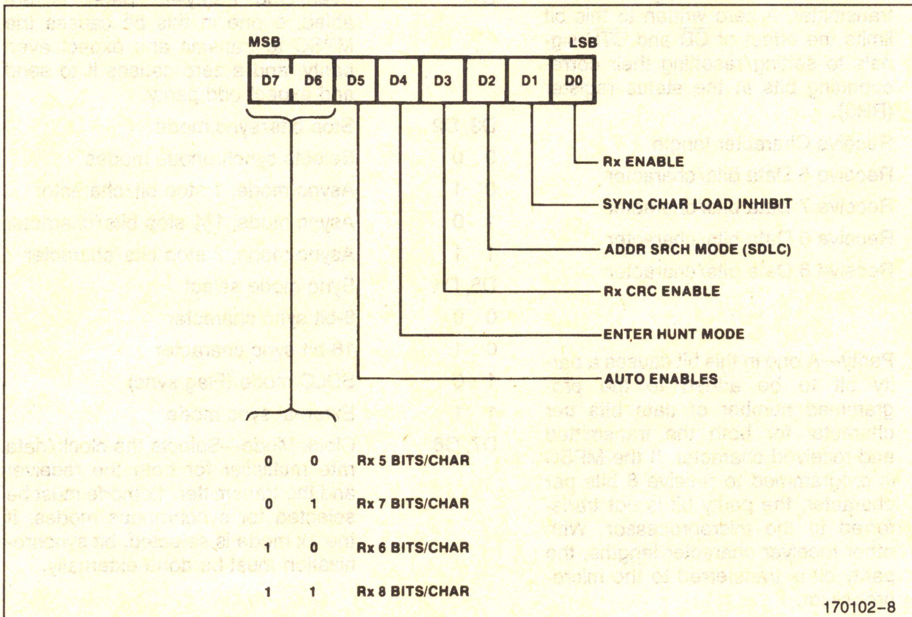


170102-7

WR2 CHANNEL B

D7–D0 Interrupt vector—This register contains the value of the interrupt vector placed on the data bus during interrupt acknowledge sequences.

Write Register 3 (WR3):



170102-8

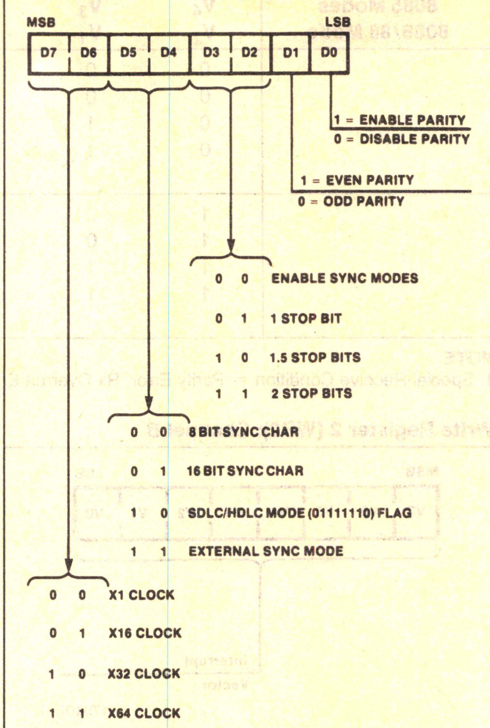
WR3

D0	Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D1	Sync Character Load Inhibit—A one prevents the receiver from loading sync characters into the receive buffers. In SDLC, this bit must be zero.
D2	Address Search Mode—If the SDLC mode has been selected, the MPSC will receive all frames unless this bit is a 1. If this bit is a 1, the MPSC will receive only frames with address (0FFH) or the value loaded into WR6. This bit must be zero in non-SDLC modes.
D3	Receive CRC Enable—A one in this bit enables (or re-enables) CRC calculation. CRC calculation starts with the last character placed in the Receiver FIFO. A zero in this bit disables, but does not reset, the Receiver CRC generator.
D4	Enter Hunt Phase—After initialization, the MPSC automatically enters the Hunt mode. If synchronization is lost, the Hunt phase can be re-entered by writing a one to this bit.
D5	Auto Enable—A one written to this bit causes $\overline{CD}$ to be automatic enable signal for the receiver and $\overline{CTS}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).
D7, D6	Receive Character length
0 0	Receive 5 Data bits/character
0 1	Receive 7 Data bits/character
1 0	Receive 6 Data bits/character
1 1	Receive 8 Data bits/character

WR4

D0	Parity—A one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

Write Register 4 (WR4):

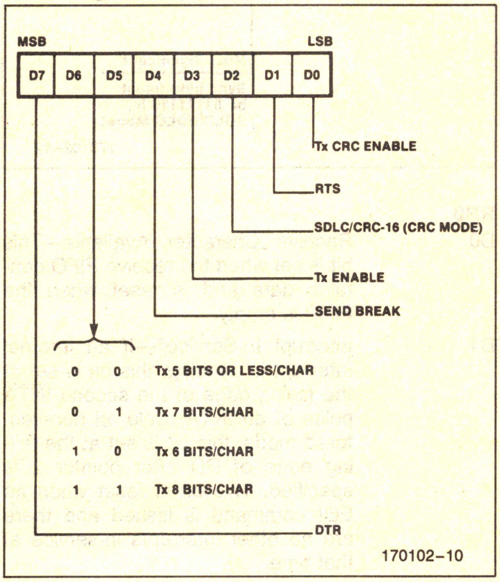


170102–9

D1	Even/Odd Parity—If parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and a zero causes it to send and expect odd parity.
D3, D2	Stop bits/sync mode
0 0	Selects synchronous modes
0 1	Async mode, 1 stop bit/character
1 0	Async mode, 1½ stop bits/character
1 1	Async mode, 2 stop bits/character
D5, D4	Sync mode select
0 0	8-bit sync character
0 1	16-bit sync character
1 0	SDLC mode (Flag sync)
1 1	External sync mode
D7, D6	Clock Mode—Selects the clock/data rate multiplier for both the receiver and the transmitter. 1x mode must be selected for synchronous modes. If the 1x mode is selected, bit synchronization must be done externally.

0 0	Clock rate = Data rate × 1
0 1	Clock rate = Data rate × 16
1 0	Clock rate = Data rate × 32
1 1	Clock rate = Data rate × 64

Write Register 5 (WR5):



170102-10

WR5

D0 Transmit CRC Enable—A one in this bit enables the transmitter CRC generator. The CRC calculation is done when a character is moved from the transmit buffer into the shift register. A zero in this bit disables CRC calculations. If this bit is not set when a transmitter underrun occurs, the CRC will not be sent.

D1 Request to Send—A one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high).

D2 CRC Select—A one in this bit selects the CRC−16 polynomial ($X^{16} + X^{15} + X^2 + 1$) and a zero in this bit selects the CCITT-CRC polynomial ($X^{16} + X^{12} + X^5 + 1$). In SDLC mode, CCITT-CRC must be selected.

D3 Transmitter Enable—A zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

D4 Send Break—A one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

D6, D5	Transmit Character length
0 0	Transmit 1–5 bits/character
0 1	Transmit 7 bits/character
1 0	Transmit 6 bits/character
1 1	Transmit 8 bits/character

Bits to be sent must be right justified least significant bit first, e.g.:

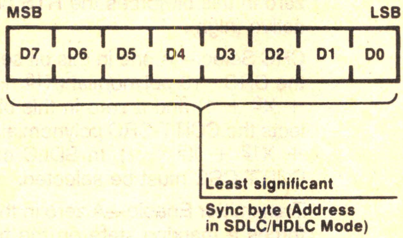
D7	D6	D5	D4	D3	D2	D1	D0
0	0	B5	B4	B3	B2	B1	B0

D7 Data Terminal Ready—When set, this bit forces the $\overline{DTR}$ pin active (low). When reset, this bit forces the $\overline{DTR}$ pin inactive (high).

Five or less mode allows transmission of one to five bits per character. The microprocessor must format the data in the following way:

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	1	1	0	0	0	B0	Sends one data bit
1	1	1	0	0	0	B1	B0	Sends two data bits
1	1	0	0	0	B2	B1	B0	Sends three data bits
1	0	0	0	B3	B2	B1	B0	Sends four data bits
0	0	0	B4	B3	B2	B1	B0	Sends five data bits

Write Register 6 (WR6):



170102-11

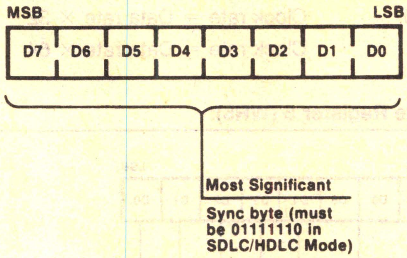
WR6

D7–D0 Sync/Address—This register contains the transmit sync character in Monosync mode, the low order 8 sync bits in Bisync mode, or the Address byte in SDLC mode.

WR7

D7–D0 Sync/Flag—This register contains the receive sync character in Monosync mode, the high order 8 sync bits in Bisync mode, or the Flag character (01111110) in SDLC mode. WR7 is not used in External Sync mode.

Write Register 7 (WR7):



170102-12

RR0

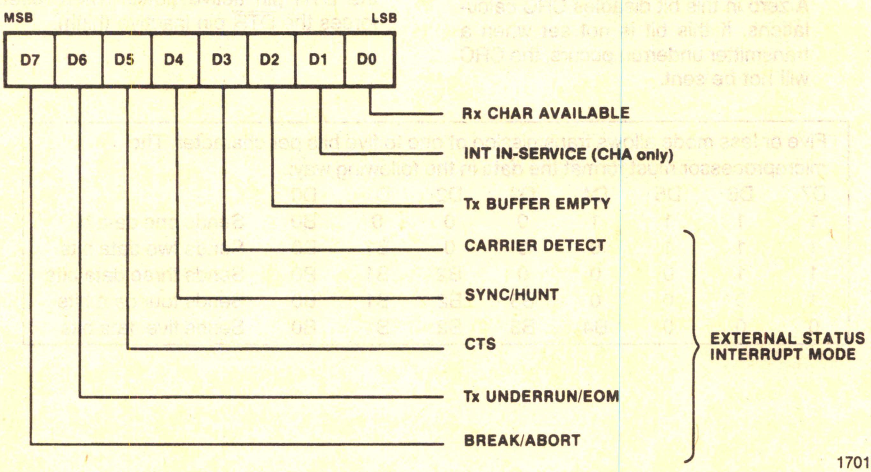
D0 Receive Character Available—This bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt In-Service*—If an Internal Interrupt is pending, this bit is set at the falling edge of the second $\overline{\text{INTA}}$ pulse of an INTA cycle. In non-vectored mode, this bit is set at the falling edge of $\overline{\text{RD}}$ after pointer 2 is specified. This bit is reset when an EOI command is issued and there are no other interrupts in-service at that time.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is

*This bit is only valid when $\overline{\text{IPI}}$ is active low and is always zero in Channel B.

Read Register 0 (RR0):



170102-13

empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the $\overline{\text{CD}}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{\text{CD}}$ pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{\text{CD}}$ pin immediately following a Reset External/Status Interrupt command.

D4 Sync/Hunt—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that Sync/Hunt shows the state of the $\overline{\text{SYNDET}}$ input. Any High-to-Low transition on the $\overline{\text{SYNDET}}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{\text{SYNDET}}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{\text{SYNDET}}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{\text{SYNDET}}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{\text{SYNDET}}$ input holds the Sync/Hunt status in the reset condition.

When external synchronization is achieved, $\overline{\text{SYNDET}}$ must be driven Low on the second rising edge of $\overline{\text{RxC}}$ after the rising edge of $\overline{\text{RxC}}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activitate the $\overline{\text{SYNDET}}$ input. Once $\overline{\text{SYNDET}}$ is forced Low, it is good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. The High-to-Low transition of the $\overline{\text{SYNDET}}$ output sets the Sync/Hunt bit, which sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt Command.

When the $\overline{\text{SYNDET}}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the MPSC again looks for a High-to-Low transition on the $\overline{\text{SYNDET}}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the MPSC is waiting for $\overline{\text{SYNDET}}$ to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the MPSC establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the MPSC to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status Interrupt, which must also be cleared by the Reset External/Status Interrupt Command. Note that the $\overline{\text{SYNDET}}$ pin acts as an output in this mode, and goes low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the MPSC. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The MPSC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

D5 — Clear to Send—This bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D6 — Transmitter Underrun/End of Message—This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set, which causes the External/Status Interrupt which must be reset by issuing a Reset External/Status command (WR0; command 2).

D7 — Break/Abort—In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

D0 — All Sent—This bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

RR1:

D3, D2, D1 — Residue Codes—Bit synchronous protocols allow I-fields that are not an integral number of characters. Since transfers from the MPSC to the CPU are character oriented, the residue codes provide the capability of receiving leftover bits. Residue bits are right justified in the last data byte received or first CRC byte.

D4 — Parity Error—If parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

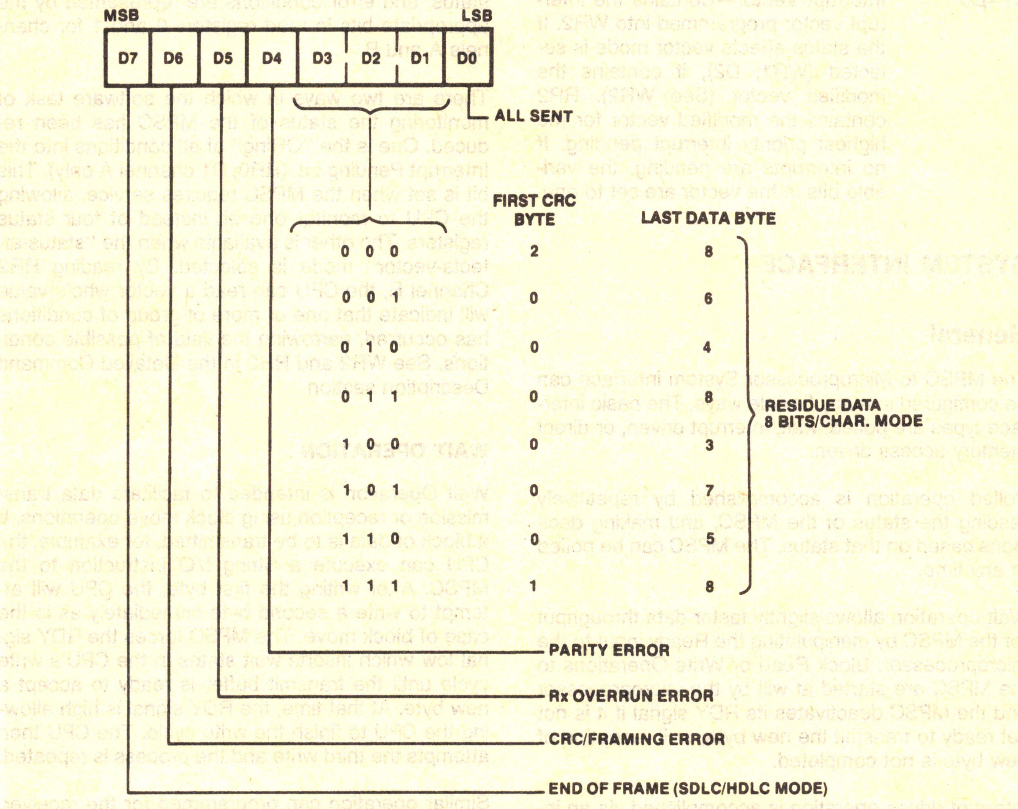
D5 — Receive Overrun Error—This bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the status affects vector mode, the overrun causes a special Receive Condition Vector.

D6 — CRC/Framing Error—In async modes, a one in this bit indicates a receive framing error. In synchronous modes, a one in this bit indicates that the calculated CRC value does not match the last two bytes received. It can be reset by issuing an Error Reset command.

SDLC Residue Code Table (I Field Bits in 2 Previous Bytes)

RR1			8 bits/char		7 bits/char		6 bits/char		5 bits/char	
D3	D2	D1	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte	First CRC Byte	Last Data Byte
1	0	0	0	3	0	2	0	1	0	5
0	1	0	0	4	0	3	0	2	0	1
1	1	0	0	5	0	4	0	3	0	2
0	0	1	0	6	0	5	0	4	0	3
1	0	1	0	7	0	6	0	5	—	—
0	1	1	0	8	0	—	—	—	—	—
1	1	1	1	8	—	—	—	—	—	—
0	0	0	2	8	1	7	0	6	0	4

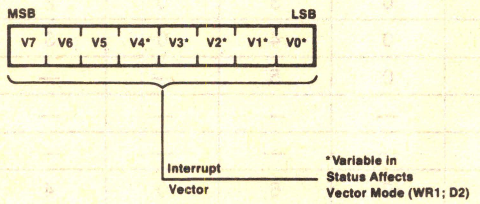
Read Register 1 (RR1): (Special Receive Condition Mode)



170102–14

D7 — End of Frame—This bit is valid only in SDLC mode. A one indicates that a valid ending flag has been received. This bit is reset either by an Error Reset command or upon reception of the first character of the next frame.

Read Register 2 (RR2):



170102-15

RR2	Channel B
D7–D0	Interrupt Vector—Contains the interrupt vector programmed into WR2. If the status affects vector mode is selected (WR1; D2), it contains the modified vector (See WR2). RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

SYSTEM INTERFACE

General

The MPSC to Microprocessor System interface can be configured in many flexible ways. The basic interface types are polled, wait, interrupt driven, or direct memory access driven.

Polled operation is accomplished by repetitively reading the status of the MPSC, and making decisions based on that status. The MPSC can be polled at any time.

Wait operation allows slightly faster data throughput for the MPSC by manipulating the Ready input to the microprocessor. Block Read or Write Operations to the MPSC are started at will by the microprocessor and the MPSC deactivates its RDY signal if it is not yet ready to transmit the new byte, or if reception of new byte is not completed.

Interrupt driven operation is accomplished via an internal or external interrupt controller. When the MPSC requires service, it sends an interrupt request signal to the microprocessor, which responds with an interrupt acknowledge signal. When the internal or external interrupt controller receives the acknowledge, it vectors the microprocessor to a service routine, in which the transaction occurs.

DMA operation is accomplished via an external DMA controller. When the MPSC needs a data transfer, it requests a DMA cycle from the DMA controller. The DMA controller then takes control of the bus and simultaneously does a read from the MPSC and a write to memory or vice-versa.

The following section describes the many configurations of these basic types of system interface techniques for both serial channels.

POLLED OPERATION

In the polled mode, the CPU must monitor the desired conditions within the MPSC by reading the appropriate bits in the read registers. All data available, status, and error conditions are represented by the appropriate bits in read registers 0 and 1 for channels A and B.

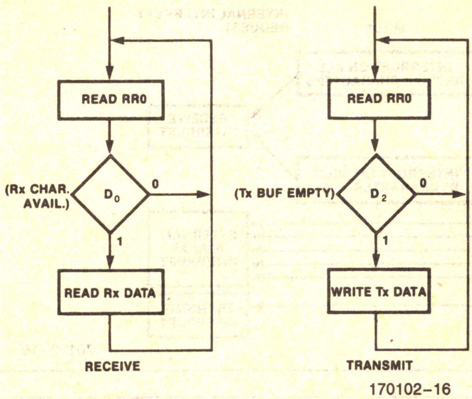
There are two ways in which the software task of monitoring the status of the MPSC has been reduced. One is the "ORing" of all conditions into the Interrupt Pending bit. (RR0; D1 channel A only). This bit is set when the MPSC requires service, allowing the CPU to monitor one bit instead of four status registers. The other is available when the "status-affects-vector" mode is selected. By reading RR2 Channel B, the CPU can read a vector who's value will indicate that one or more of group of conditions has occurred, narrowing the field of possible conditions. See WR2 and RR2 in the Detailed Command Description section.

WAIT OPERATION

Wait Operation is intended to facilitate data transmission or reception using block move operations. If a block of data is to be transmitted, for example, the CPU can execute a String I/O instruction to the MPSC. After writing the first byte, the CPU will attempt to write a second byte immediately as is the case of block move. The MPSC forces the RDY signal low which inserts wait states in the CPU's write cycle until the transmit buffer is ready to accept a new byte. At that time, the RDY signal is high allowing the CPU to finish the write cycle. The CPU then attempts the third write and the process is repeated.

Similar operation can programmed for the receiver. During initialization, wait on transmit (WR1; D5 = 0)

Software Flow, Polled Operation



170102–16

NOTES:
1. RR0; D0 is reset automatically when the data is read.
2. RR0; D2 is reset automatically when the data is written.

or wait on receive (WR1; D5 = 1) can be selected. The wait operation can be enabled/disabled by setting/resetting the Wait Enable Bit (WR1; D7).

NOTE:
CAUTION: ANY CONDITION THAT CAN CAUSE THE TRANSMITTER TO STOP (E.G., CTS GOES INACTIVE) OR THE RECEIVER TO STOP (E.G., RX DATA STOPS) WILL CAUSE THE MPSC TO HANG THE CPU UP IN WAIT STATES UNTIL RESET. EXTREME CARE SHOULD BE TAKEN WHEN USING THIS FEATURE.

INTERRUPT DRIVEN OPERATION

The MPSC can be programmed into several interrupt modes: Non-Vectored, 8085 vectored, and 8088/86 vectored. In both vectored modes, multiple MPSC's can be daisy-chained.

In the vectored mode, the MPSC responds to an interrupt acknowledge sequence by placing a call instruction (8085 mode) and interrupt vector (8085 and 8088/86 mode) on the data bus.

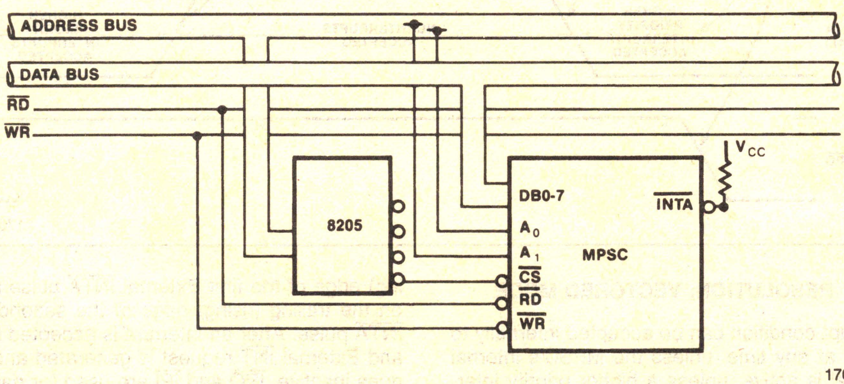
The MPSC can be programmed to cause an interrupt due to up to 14 conditions in each channel. The status of these interrupt conditions is contained in Read Registers 0 and 1. These 14 conditions are all directed to cause 3 different types of internal interrupt request for each channel: receive/interrupts, transmit interrupts and external/status interrupts (if enabled).

This results in up to 6 internal interrupt request signals. The priority of those signals can be programmed to one of two fixed modes:

Highest Priority					Lowest Priority
RxA	RxB	TxA	TxB	ExTA	ExTB
RxA	TxA	RxB	TxB	ExTA	ExTB

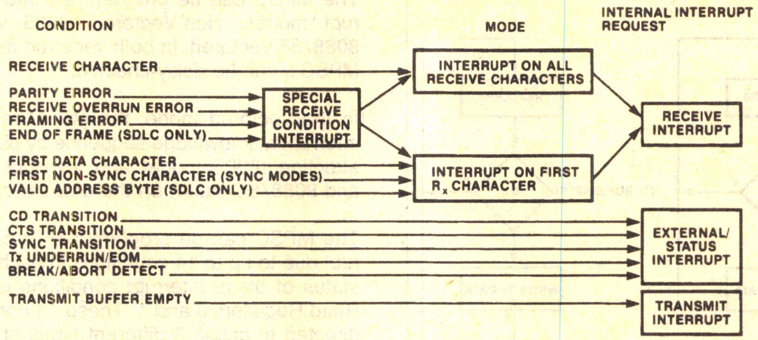
The interrupt priority resolution works differently for vectored and non-vectored modes.

Hardware Configuration, Polled Operation

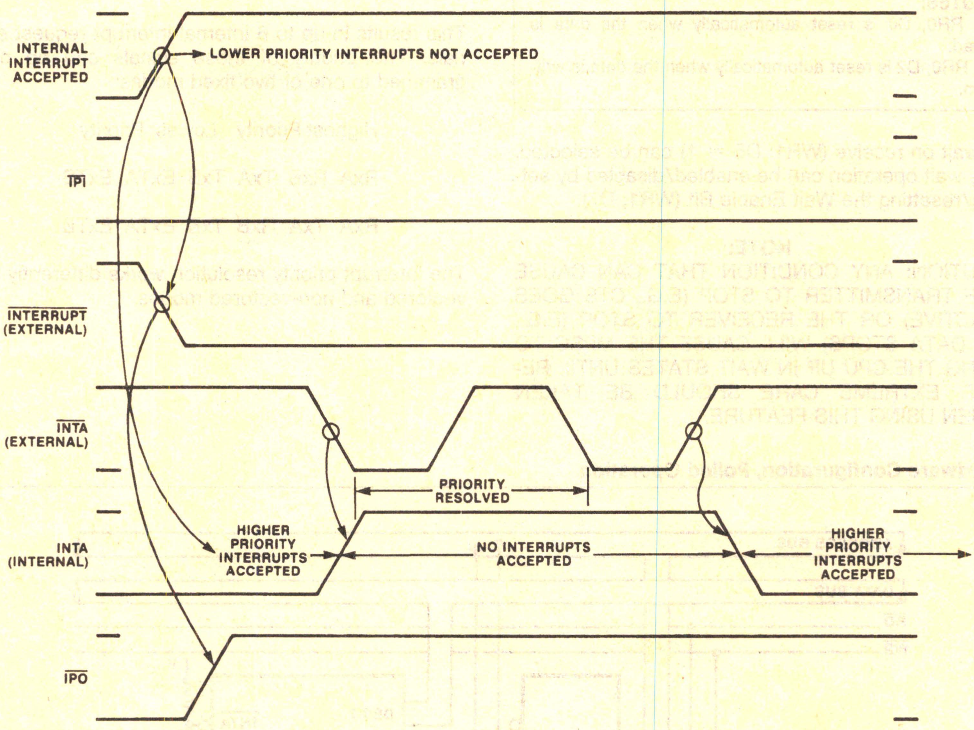


170102–17

Interrupt Condition Grouping



170102-18

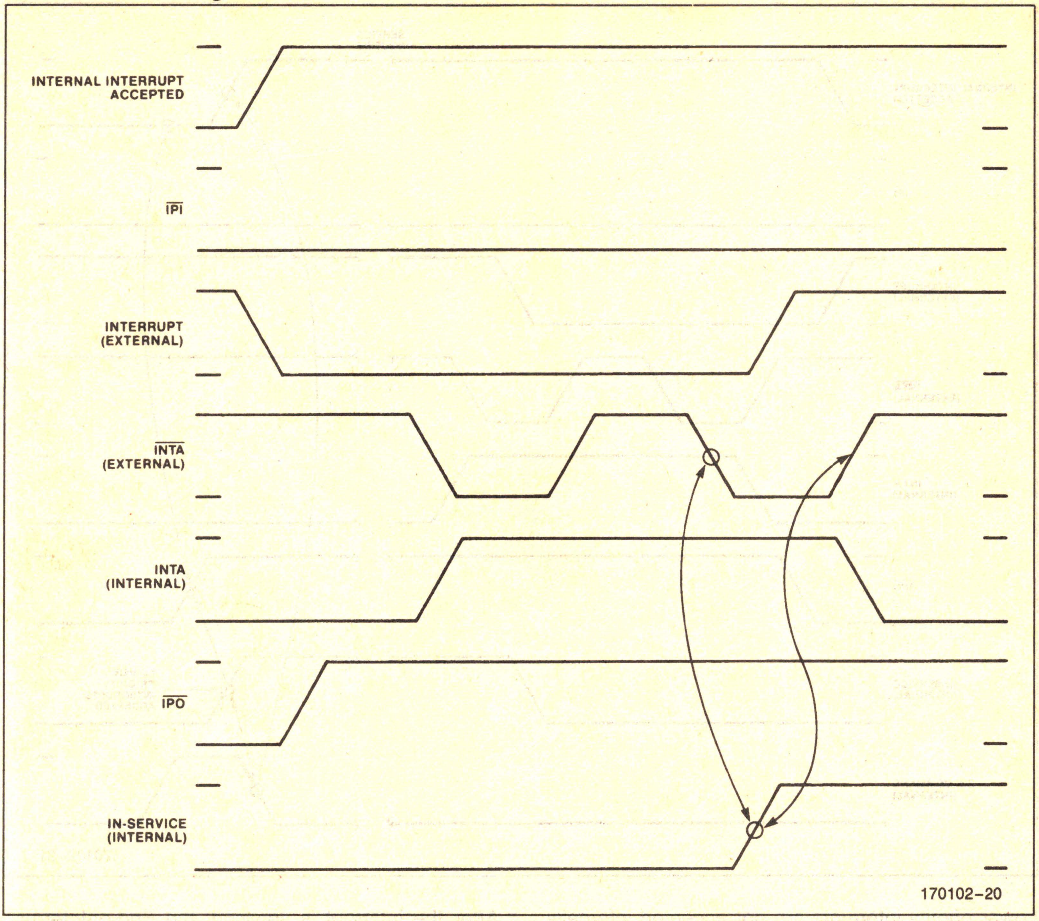


170102-19

PRIORITY RESOLUTION: VECTORED MODE

Any interrupt condition can be accepted internally to the MPSC at any time, unless the MPSC's internal INTA signal is active, unless a higher priority interrupt is currently accepted, or if $\overline{\text{IPI}}$ is inactive (high). The MPSC's internal INTA is set on the leading (falling) edge of the first External $\overline{\text{INTA}}$ pulse and reset on the trailing (rising) edge of the second External $\overline{\text{INTA}}$ pulse. After an interrupt is accepted internally, and External $\overline{\text{INT}}$ request is generated and the $\overline{\text{IPO}}$ goes inactive. $\overline{\text{IPO}}$ and $\overline{\text{IPI}}$ are used for daisy-chaining MPSC's together.

In-Service Timing

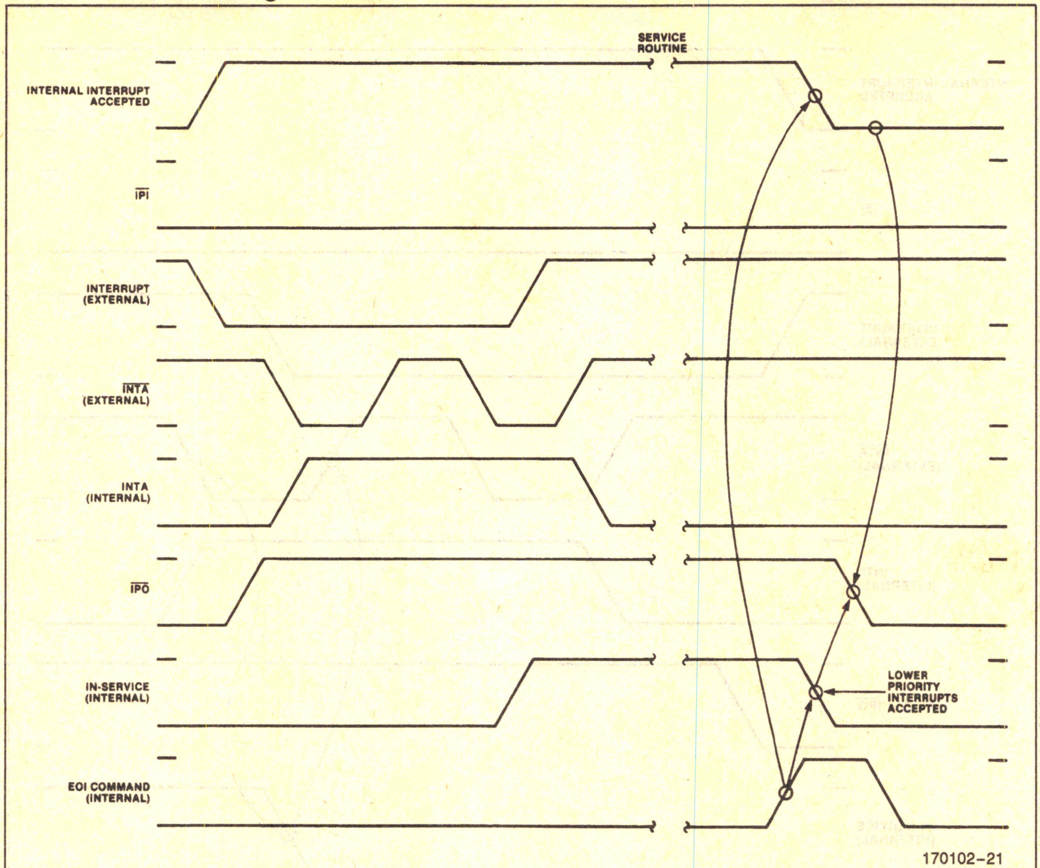


The MPSC's internal INTA is set on the leading (falling) edge of the first external $\overline{\text{INTA}}$ pulse, and reset on the trailing (rising) edge of the second external $\overline{\text{INTA}}$ pulse. After an interrupt is accepted internally, and external $\overline{\text{INT}}$ request is generated and $\overline{\text{IPO}}$ goes inactive (high). $\overline{\text{IPO}}$ and $\overline{\text{IPI}}$ are used for daisy-chaining MPSC's together.

Each of the six interrupt sources has an associated In-Service latch. After priority has been resolved, the highest priority In-Service latch is set. After the In-Service latch is set, the $\overline{\text{INT}}$ pin goes inactive (high).

NOTE:
If the External $\overline{\text{INT}}$ pin is active and the IPI signal is pulled inactive high, the $\overline{\text{INT}}$ signal will also go inactive. $\overline{\text{IPI}}$ qualifies the External $\overline{\text{INT}}$ Signal.

EOI Command Timing

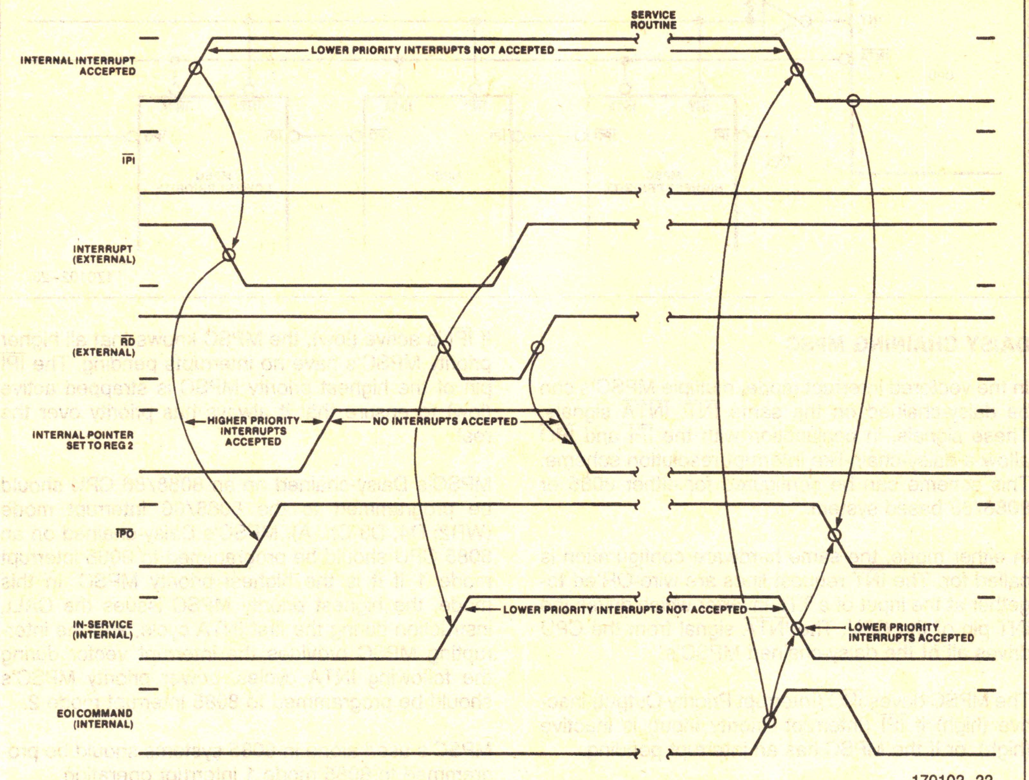


170102-21

Lower priority interrupts are not accepted internally while the In-Service latch is set. However, higher priority interrupts are accepted internally and a new external $\overline{INT}$ request is generated. If the CPU responds with a new INTA sequence, the MPSC will respond as before, suspending the lower priority interrupt.

After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC. This command will cause an internal pulse that is used to reset the In-Service Latch which allows service for lower priority interrupts in the daisy-chain to resume, provided a new INTA sequence does not start for a higher priority interrupt (higher than the highest under service). If there is no interrupt pending internally, the $\overline{IPO}$ follows $\overline{IPI}$.

Non-Vectored Interrupt Timing



170102–22

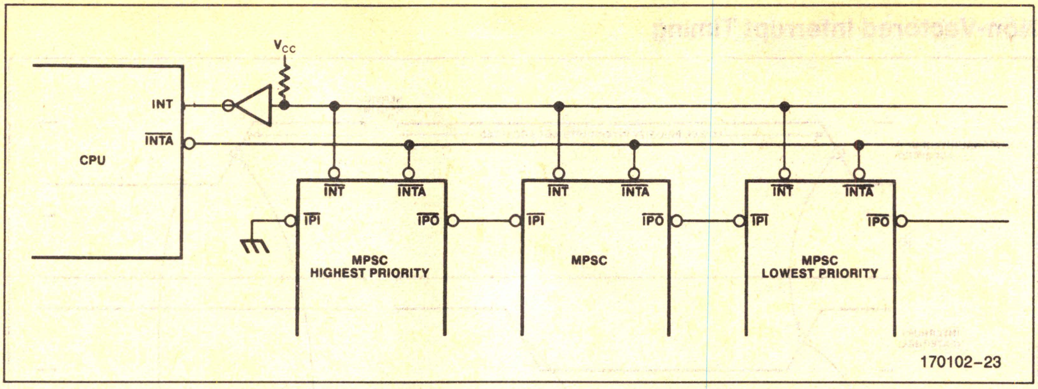
PRIORITY RESOLUTION: NON-VECTORED MODE

In non-vectored mode, the MPSC does not respond to interrupt acknowledge sequences. The $\overline{\text{INTA}}$ input (pin 27) must be pulled high for proper operation. The MPSC should be programmed to the Status-Affects-Vector mode, and the CPU should read RR2 (Ch. B) in its service routine to determine which interrupt requires service.

In this case, the internal pointer being set to RR2 provides the same function as the internal INTA signal in the vectored mode. It inhibits acceptance of any additional internal interrupts and its leading edge starts the interrupt priority resolution circuit. The interrupt priority resolution is ended by the leading edge of the read signal used by the CPU to retrieve the modified vector. The leading edge of read sets the In-Service latch and forces the external $\overline{\text{INT}}$ output inactive (high). The internal pointer is reset to zero after the trailing edge of the read pulse.

NOTE:

That if RR2 is specified but not read, no internal interrupts, regardless of priority, are accepted.



DAISY CHAINING MPSC

In the vectored interrupt mode, multiple MPSC's can be daisy-chained on the same $\overline{INT}$, $\overline{INTA}$ signals. These signals, in conjunction with the $\overline{IPI}$ and $\overline{IPO}$ allow a daisy-chain-like interrupt resolution scheme. This scheme can be configured for either 8085 or 8086/88 based system.

In either mode, the same hardware configuration is called for. The $\overline{INT}$ request lines are wire-OR'ed together at the input of a TTL inverter which drives the INT pin of the CPU. The $\overline{INTA}$ signal from the CPU drives all of the daisy-chained MPSC's.

The MPSC drives $\overline{IPO}$ (Interrupt Priority Output) inactive (high) if $\overline{IPI}$ (Interrupt Priority Input) is inactive (high), or if the MPSC has an interrupt pending.

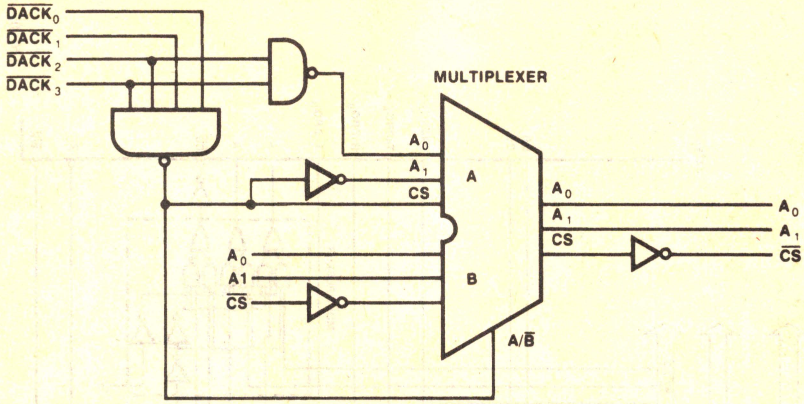
The $\overline{IPO}$ of the highest priority MPSC is connected to the $\overline{IPI}$ of the next highest priority MPSC, and so on.

If $\overline{IPI}$ is active (low), the MPSC knows that all higher priority MPSC's have no interrupts pending. The $\overline{IPI}$ pin of the highest priority MPSC is strapped active (low) to ensure that it always has priority over the rest.

MPSC's Daisy-chained on an 8088/86 CPU should be programmed to the 8088/86 Interrupt mode (WR2; D4, D3 Ch. A). MPSC's Daisy-chained on an 8085 CPU should be programmed to 8085 interrupt mode 1 if it is the highest priority MPSC. In this mode, the highest priority MPSC issues the CALL instruction during the first INTA cycle, and the interrupting MPSC provides the interrupt vector during the following INTA cycles. Lower priority MPSC's should be programmed to 8085 interrupt mode 2.

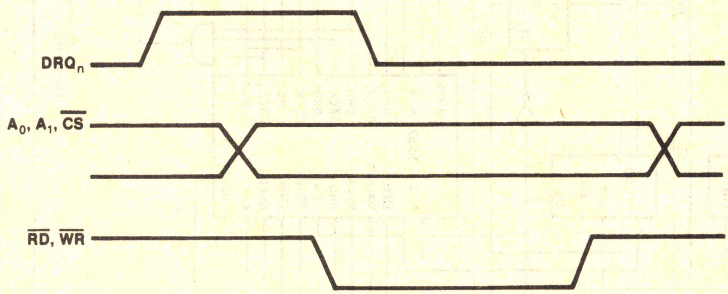
MPSC's used alone in 8085 systems should be programmed to 8085 mode 1 interrupt operation.

DMA Acknowledge Circuit



170102-24

DMA Timing



170102-25

DMA OPERATION

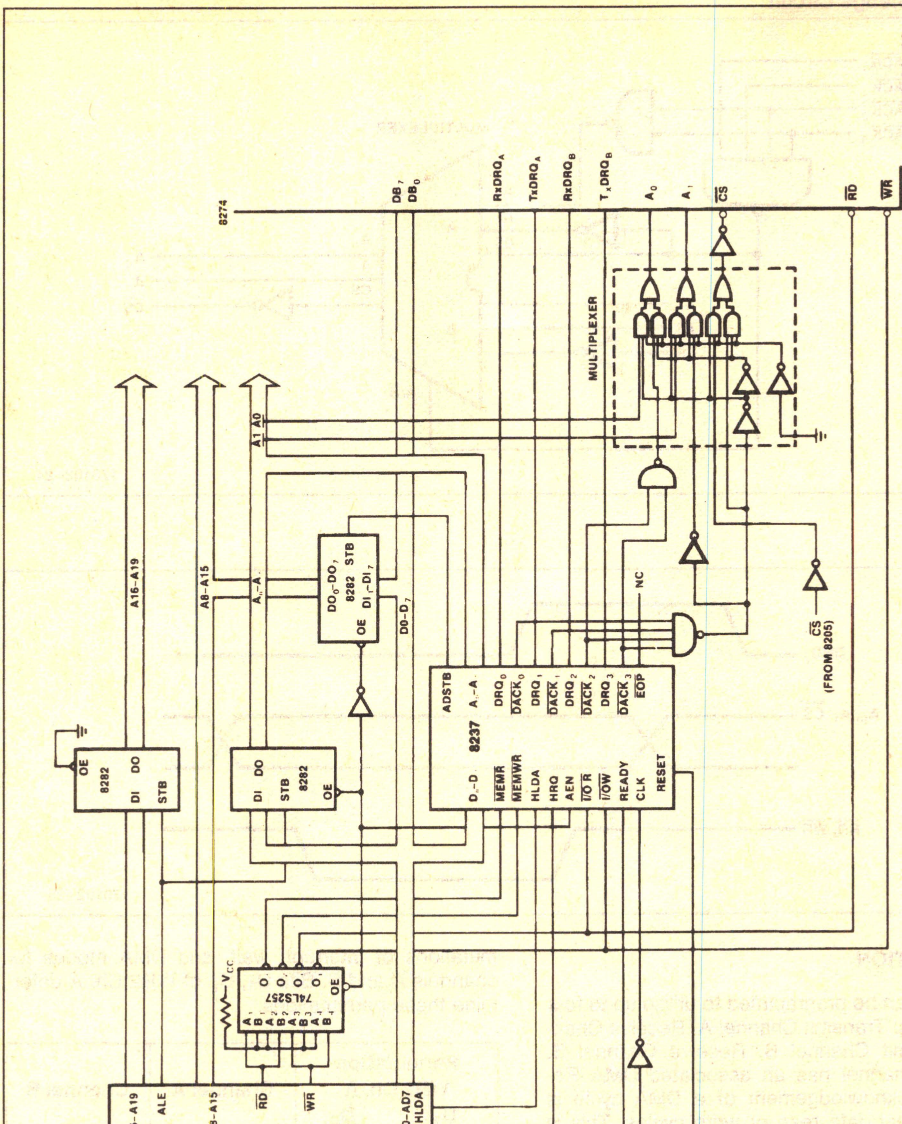
Each MPSC can be programmed to utilize up to four DMA channels: Transmit Channel A, Receive Channel A, Transmit Channel B, Receive Channel B. Each DMA Channel has an associated DMA Request line. Acknowledgement of a DMA cycle is done via normal data read or write cycles. This is accomplished by encoding the $\overline{DACK}$ signals to generate A_0, A_1, and $\overline{CS}$, and multiplexing them with the normal A_0, A_1, and $\overline{CS}$ signals.

PERMUTATIONS

Channels A and B can be used with different system interface modes. In all cases it is possible to poll the MPSC. The following table shows the possible permutations of interrupt, wait, and DMA modes for channels A and B. Bits D_1, D_0 of WR2 Ch. A determine these permutations.

Permutation WR2 Ch. A D_1	D_0	Channel A	Channel B
0	0	Wait Interrupt Polled	Wait Interrupt Polled
0	1	DMA Polled	Interrupt Polled
1	0	DMA Polled	DMA Polled

NOTE:
D1, D0 = 1, 1 is illegal.



170102-26

NOTE:
The circuit was not designed based on a worst-case timing analysis. Specific implementations should include this timing analysis.

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. The CRC bytes are not to be used for CRC verification. Residue bits may be contained in the first CRC byte. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1: D6) and residue code (RR1; D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit under-run/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3; D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
Under Bias ....0°C to +70°C
Storage Temperature
(Ceramic Package) ....−65°C to +150°C
(Plastic Package) ....−40°C to +125°C
Voltage on Any Pin with
Respect to Ground ....−0.5V to +7.0V

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to +70°C; V_{CC} = +5V ±10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	+2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		+0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = 200 μA
I_{IL}	Input Leakage Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		200	mA	

NOTE:
1. For Extended Temperature EXPRESS, use MIL8274 electrical Parameters.

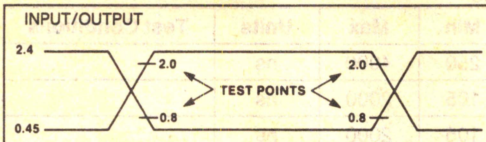
CAPACITANCE T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
C_{OUT}	Output Capacitance		15	pF	Unmeasured pins returned to GND
$C_{I/O}$	Input/Output Capacitance		20	pF	

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$; $V_{CC} = +5V \pm 10\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{CY}	CLK Period	250	4000	ns	
t_{CL}	CLK Low Time	105	2000	ns	
t_{CH}	CLK High Time	105	2000	ns	
t_r	CLK Rise Time	0	30	ns	
t_f	CLK Fall Time	0	30	ns	
t_{AR}	A0, A1 Setup to $\overline{RD}$ ↓	0		ns	
t_{AD}	A0, A1 to Data Output Delay		200	ns	$C_L = 150$ pF
t_{RA}	A0, A1 Hold after $\overline{RD}$ ↑	0		ns	
t_{RD}	$\overline{RD}$ ↓ to Data Output Delay		200	ns	$C_L = 150$ pF
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{DF}	Output Float Delay		120	ns	
t_{AW}	$\overline{CS}$, A0, A1 Setup to $\overline{WR}$ ↓	0		ns	
t_{WA}	$\overline{CS}$, A0, A1 Hold after $\overline{WR}$ ↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$ ↑	150		ns	
t_{WD}	Data Hold after $\overline{WR}$ ↑	0		ns	
t_{PI}	$\overline{IPI}$ Setup to $\overline{INTA}$ ↓	0		ns	
t_{IP}	$\overline{IPI}$ Hold after $\overline{INTA}$ ↑	10		ns	
t_{II}	$\overline{INTA}$ Pulse Width	250		ns	
t_{PIPO}	$\overline{IPI}$ ↓ to $\overline{IPO}$ Delay		100	ns	
t_{ID}	$\overline{INTA}$ ↓ to Data Output Delay		200	ns	
t_{CQ}	$\overline{RD}$ or $\overline{WR}$ to DRQ ↓		150	ns	
t_{RV}	Recovery Time Between Controls	300		ns	
t_{CW}	$\overline{CS}$, A0, A1 to RDY_A or RDY_B Delay		140	ns	
t_{DCY}	Data Clock Cycle	4.5		tcy	
t_{DCL}	Data Clock Low Time	180		ns	
t_{DCH}	Data Clock High Time	180		ns	
t_{TD}	$\overline{TxC}$ to TxD Delay (x1 Mode)		300	ns	
t_{DS}	RxD Setup to $\overline{RxC}$ ↑	0		ns	
t_{DH}	RxD Hold after $\overline{RxC}$ ↑	140		ns	
t_{ITD}	$\overline{TxC}$ to $\overline{INT}$ Delay	4	6	tcy	
t_{IRD}	RxC to $\overline{INT}$ Delay	7	10	tcy	
t_{PL}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ Low Time	200		ns	
t_{PH}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ High Time	200		ns	
t_{IPD}	External $\overline{INT}$ from $\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$		500	ns	

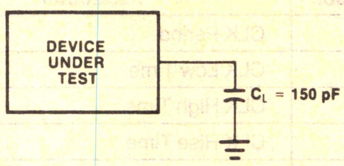
A.C. TESTING INPUT/OUTPUT WAVEFORM



170102-27

A.C. Testing; Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

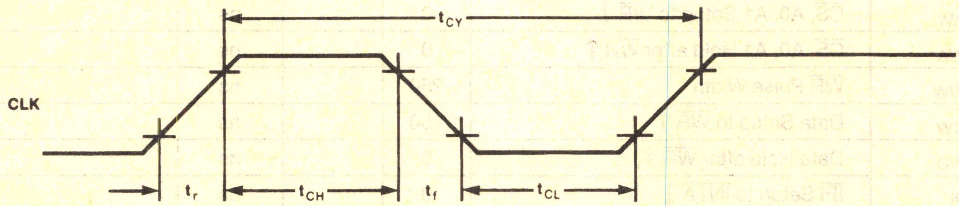


170102-28

C_L = 100 pF
C_L Includes Jig Capacitance

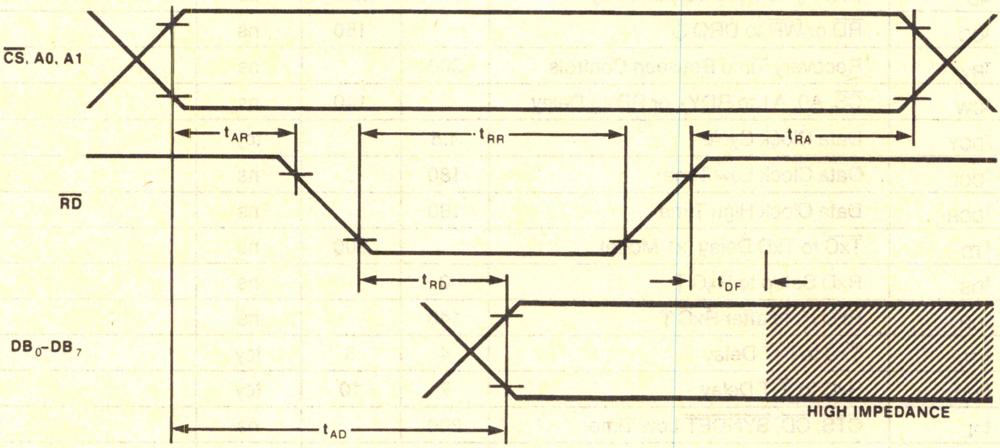
WAVEFORMS

CLOCK CYCLE



170102-29

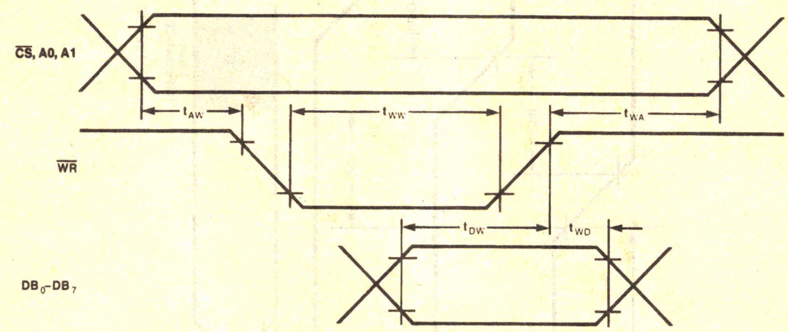
READ CYCLE



170102-30

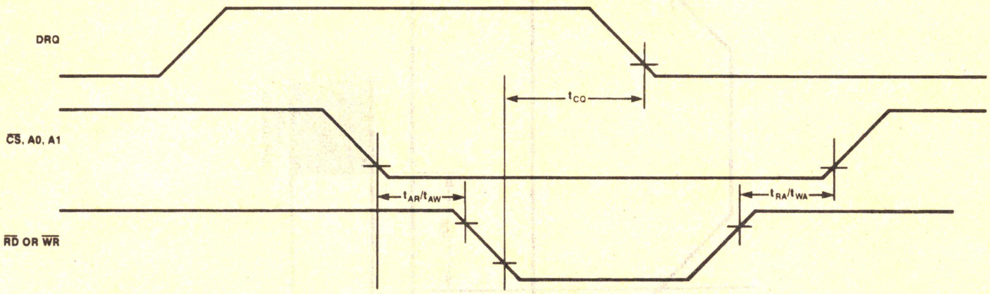
WAVEFORMS (Continued)

WRITE CYCLE



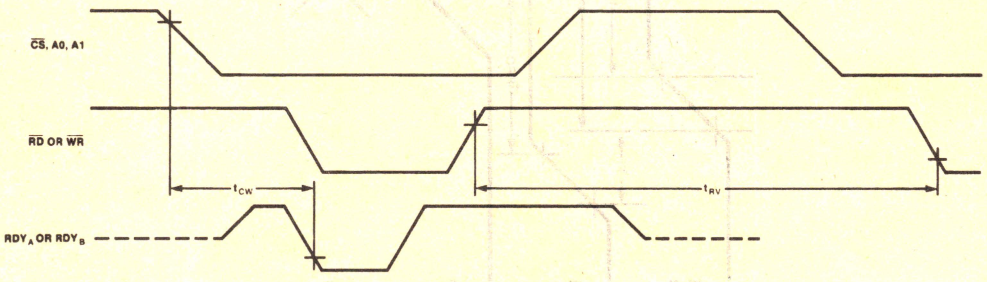
170102-31

DMA CYCLE



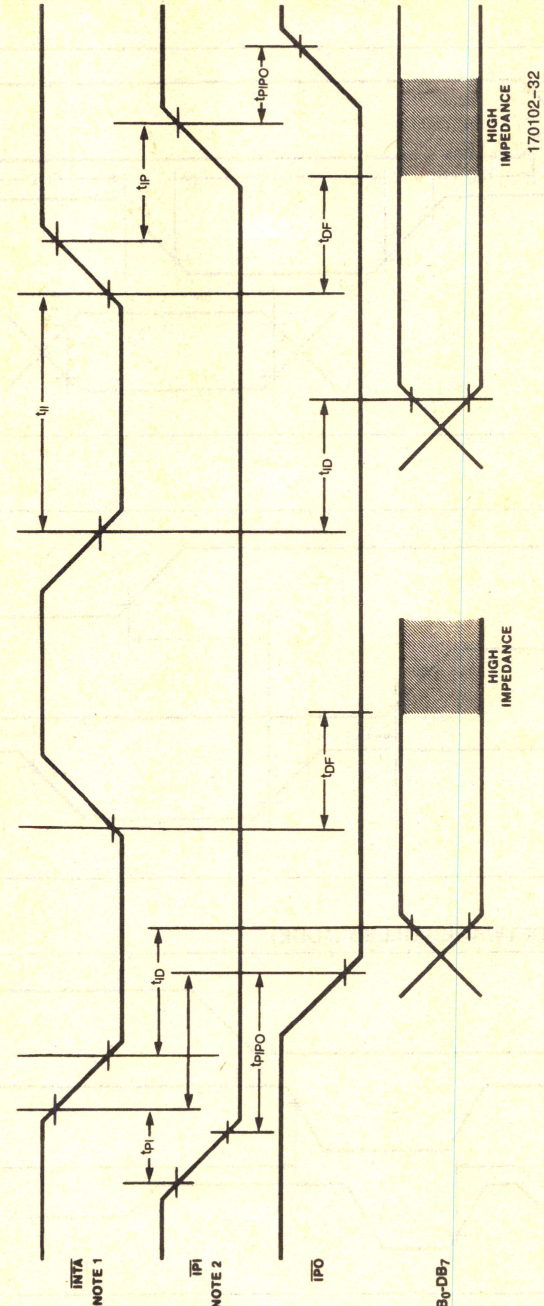
170102-33

READ/WRITE CYCLE (SOFTWARE POLLED MODE)



170102-34

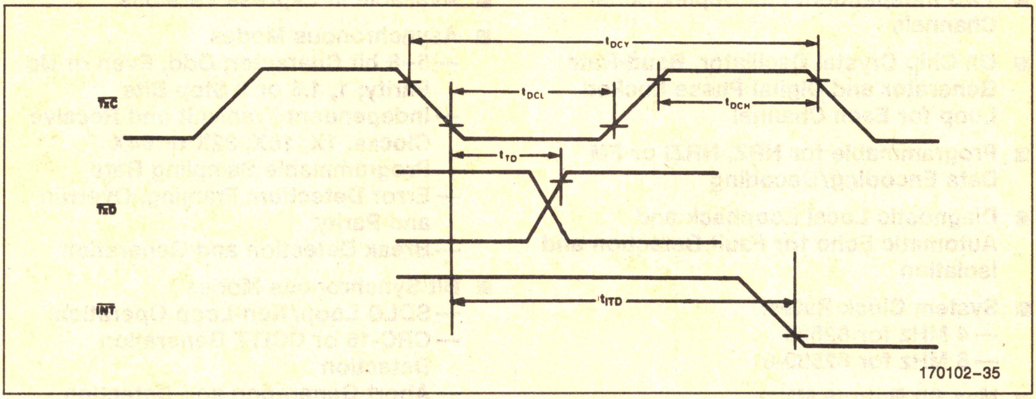
INTA CYCLE



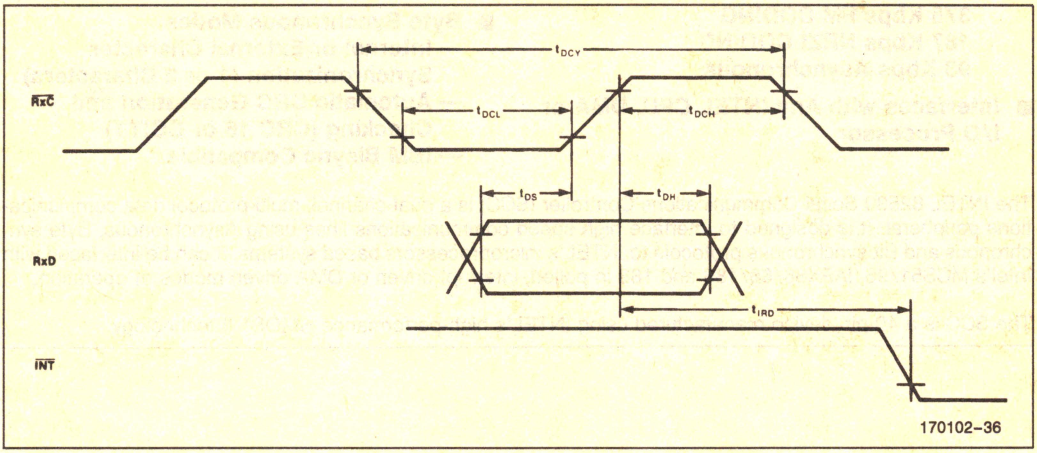
NOTES:
1. $\overline{\text{INTA}}$ signal as $\overline{\text{RD}}$ signal.
2. $\overline{\text{IPI}}$ signal acts as $\overline{\text{CS}}$ signal.

WAVEFORMS (Continued)

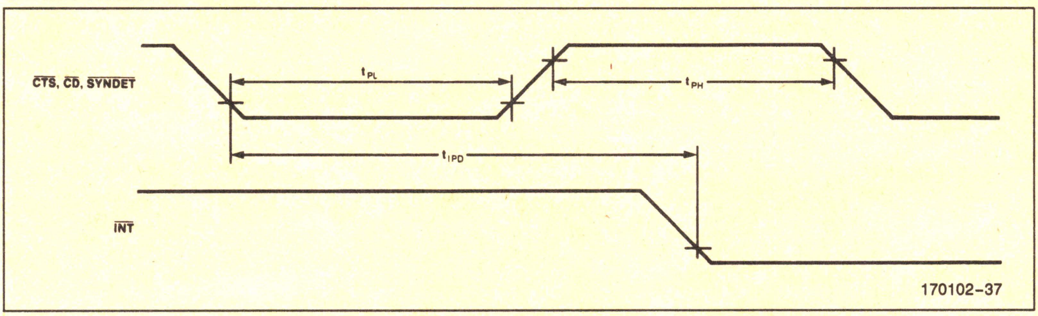
TRANSMIT DATA CYCLE



RECEIVE DATA CYCLE



OTHER TIMING



intel®

82530/82530-6
SERIAL COMMUNICATIONS CONTROLLER (SCC)

- Two Independent Full Duplex Serial Channels
- On Chip Crystal Oscillator, Baud-Rate Generator and Digital Phase Locked Loop for Each Channel
- Programmable for NRZ, NRZI or FM Data Encoding/Decoding
- Diagnostic Local Loopback and Automatic Echo for Fault Detection and Isolation
- System Clock Rates:
  - 4 MHz for 82530
  - 6 MHz for 82530-6
- Max Bit Rate (6 MHz)
  - Externally Clocked: 1.5 Mbps
    Self-Clocked:
    375 Kbps FM CODING
    187 Kbps NRZI CODING
    93 Kbps Asynchronous
- Interfaces with Any INTEL CPU, DMA or I/O Processor
- Available in Express Versions
- Asynchronous Modes
  - 5–8 bit Character; Odd, Even or No Parity; 1, 1.5 or 2 Stop Bits
  - Independent Transmit and Receive Clocks. 1X, 16X, 32X or 64X Programmable Sampling Rate
  - Error Detection: Framing, Overrun and Parity
  - Break Detection and Generation
- Bit Synchronous Modes
  - SDLC Loop/Non-Loop Operation
  - CRC-16 or CCITT Generation Detection
  - Abort Generation and Detection
  - I-field Residue Handling
  - CCITT X.25 Compatible
- Byte Synchronous Modes
  - Internal or External Character Synchronization (1 or 2 Characters)
  - Automatic CRC Generation and Checking (CRC 16 or CCITT)
  - IBM Bisync Compatible

The INTEL 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. It is designed to interface high speed communications lines using Asynchronous, Byte synchronous and Bit synchronous protocols to INTEL's microprocessors based systems. It can be interfaced with Intel's MCS51/96, iAPX86/88/186 and 188 in polled, interrupt driven or DMA driven modes of operation.

The SCC is a 40-pin device manufactured using INTEL's high-performance HMOS* II technology.

* HMOS is a patented process of Intel Corporation.

November 1986
Order Number: 230834-002

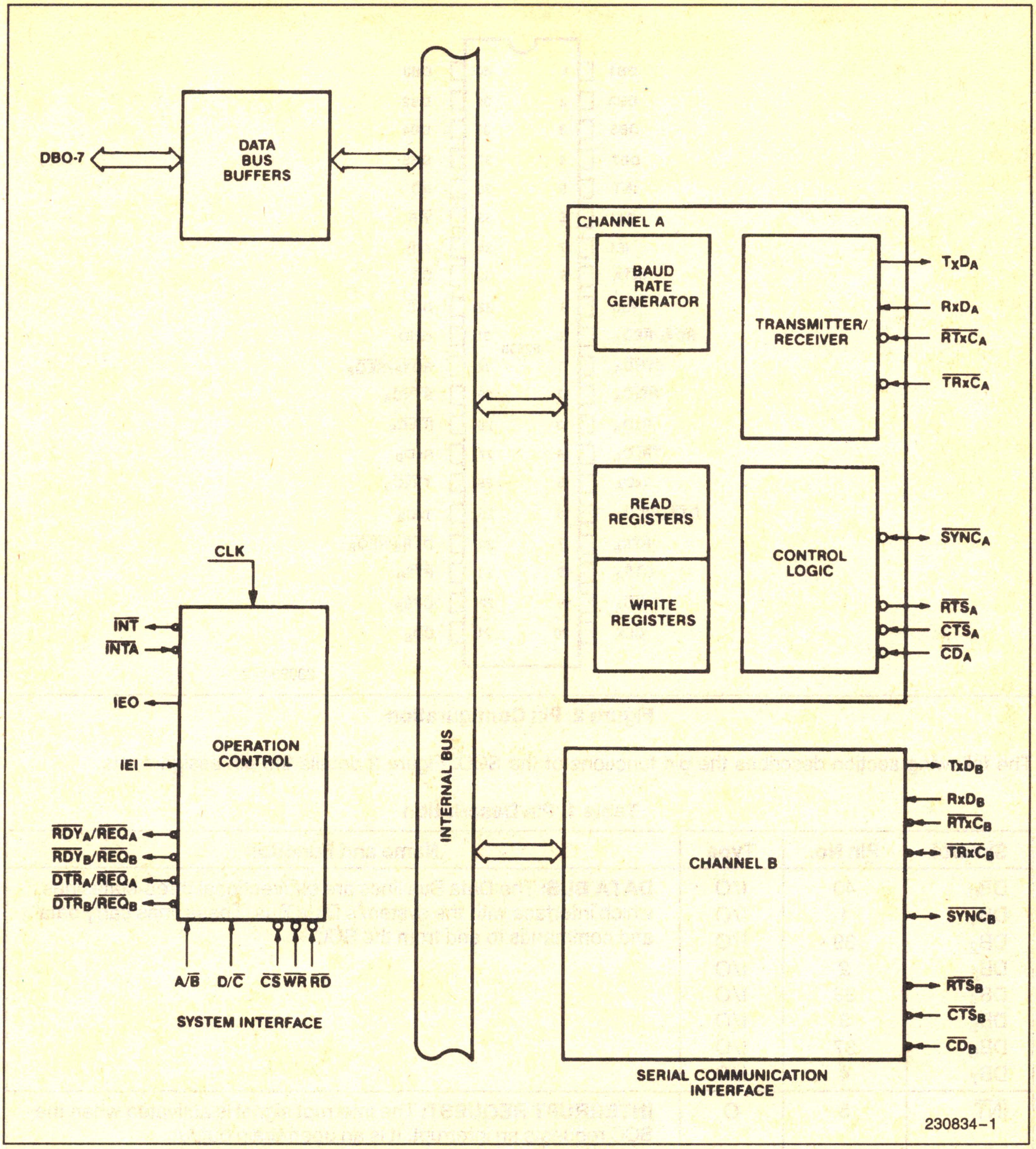


Figure 1. 82530 Internal Block Diagram

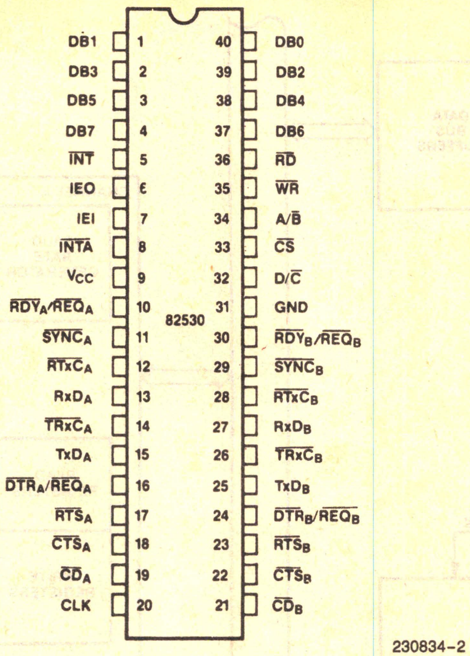


Figure 2. Pin Configuration

The following section describes the pin functions of the SCC. Figure 2 details the pin assignments.

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	40 1 39 2 38 3 37 4	I/O I/O I/O I/O I/O I/O I/O I/O	**DATA BUS:** The Data Bus lines are bi-directional three-state lines which interface with the system's Data Bus. These lines carry data and commands to and from the SCC.
$\overline{INT}$	5	O	**INTERRUPT REQUEST:** The interrupt signal is activated when the SCC requests an interrupt. It is an open drain output.
IEO	6	O	**INTERRUPT ENABLE OUT:** IEO is High only if IEI is High and the CPU is not servicing an SCC interrupt or the SCC is not requesting an interrupt (Interrupt Acknowledge cycle only). IEO is connected to the next lower priority device's IEI input and thus inhibits interrupts from lower priority devices.
IEI	7	I	**INTERRUPT ENABLE IN:** IEI is used with IEO to form an interrupt daisy chain when there is more than one interrupt-driven device. A High IEI indicates that no other higher priority device has an interrupt under service or is requesting an interrupt.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{INTA}$	8	I	**INTERRUPT ACKNOWLEDGE:** This signal indicates an active Interrupt Acknowledge cycle. During this cycle, the SCC interrupt daisy chain settles. When $\overline{RD}$ becomes active, the SCC places an interrupt vector on the data bus (if IEI is High). $\overline{INTA}$ is latched by the rising edge of CLK.
V_{CC}	9		**POWER:** +5V Power supply.
$\overline{RDY_A}/\overline{REQ_A}$ $\overline{RDY_B}/\overline{REQ_B}$	10 30	O O	**READY/REQUEST:** (output, open-drain when programmed for a Ready function, driven High or Low when programmed for a Request function). These dual-purpose outputs may be programmed as Request lines for a DMA controller or as Ready lines to synchronize the CPU to the SCC data rate. The reset state is Ready.
$\overline{SYNC_A}$ $\overline{SYNC_B}$	11 29	I/O I/O	**SYNCHRONIZATION:** These pins can act either as inputs, outputs or part of the crystal oscillator circuit. In the Asynchronous receive mode (crystal oscillator option not selected), these pins are inputs similar to $\overline{CTS}$ and $\overline{CD}$. In this mode, transitions on these lines affect the state of the Synchronous/Hunt status bits in Read Register 0 (Figure 9) but have no other function. In External Synchronization mode with the crystal oscillator not selected, these lines also act as inputs. In this mode, $\overline{SYNC}$ must be driven LOW two receive clock cycles after the last bit in the synchronous character is received. Character assembly begins on the rising edge of the receive clock immediately preceding the activation of $\overline{SYNC}$. In the Internal Synchronization mode (Monosync and Bisync) with the crystal oscillator not selected, these pins act as outputs and are active only during the part of the receive clock cycle in which synchronous characters are recognized. The synchronous condition is not latched, so these outputs are active each time a synchronization pattern is recognized (regardless of characters boundaries). In SDLC mode, these pins act as outputs and are valid on receipt of a flag.
$\overline{RTxC_A}$ $\overline{RTxC_B}$	12 28	I I	**RECEIVE/TRANSMIT CLOCKS:** These pins can be programmed in several different modes of operation. In each channel, $\overline{RTxC}$ may supply the receive clock, the transmit clock, the clock for the baud rate generator, or the clock for the Digital Phase Locked Loop. These pins can be programmed for use with the respective $\overline{SYNC}$ pins as a crystal oscillator. The receive clock may be 1, 16, 32, or 64 times the data rate in Asynchronous modes.
RxD_A RxD_B	13 27	I I	**RECEIVE DATA:** These lines receive serial data at standard TTL levels.
$\overline{TRxC_A}$ $\overline{TRxC_B}$	14 26	I/O I/O	**TRANSMIT/RECEIVE CLOCKS:** These pins can be programmed in several different modes of operation. $\overline{TRxC}$ may supply the receive clock or the transmit clock in the input mode or supply the output of the Digital Phase Locked Loop, the crystal oscillator, the baud rate generator, or the transmit clock in the output mode.
TxD_A TxD_B	15 25	O O	**TRANSMIT DATA:** These output signals transmit serial data at standard TTL levels
$\overline{DTR_A}/\overline{REQ_A}$ $\overline{DTR_B}/\overline{REQ_B}$	16 24	O O	**DATA TERMINAL READY/REQUEST:** These outputs follow the state programmed into the DTR bit. They can also be used as general purpose outputs or as Request lines for a DMA controller.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RTS}_A$ $\overline{RTS}_B$	17 23	O O	**REQUEST TO SEND:** When the Request to Send (RTS) bit in Write Register 5 is set (Figure 10), the $\overline{\text{RTS}}$ signal goes Low. When the RTS bit is reset in the Asynchronous mode and Auto Enable is on, the signal goes High after the transmitter is empty. In Synchronous mode or in Asynchronous mode with Auto Enable off, the $\overline{\text{RTS}}$ pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.
$\overline{CTS}_A$ $\overline{CTS}_B$	18 22	I I	**CLEAR TO SEND:** If these pins are programmed as Auto Enables, a Low on the inputs enables the respective transmitters. If not programmed as Auto Enables, they may be used as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow rise-time inputs. The SCC detects pulses on these inputs and can interrupt the CPU on both logic level transitions.
$\overline{CD}_A$ $\overline{CD}_B$	19 21	I I	**CARRIER DETECT:** These pins function as receiver enables if they are programmed for Auto Enables; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered to accommodate slow rise time signals. The SCC detects pulses on these pins and can interrupt the CPU on both logic level transitions.
CLK	20	I	**CLOCK:** This is the system SCC clock used to synchronize internal signals.
GND	31		**GROUND**
D/$\overline{C}$	32	I	**DATA/COMMAND SELECT:** This signal defines the type of information transferred to or from the SCC. A High means data is transferred; a Low indicates a command.
$\overline{CS}$	33	I	**CHIP SELECT:** This signal selects the SCC for a read or write operation.
A/$\overline{B}$	34	I	**CHANNEL A/CHANNEL B SELECT:** This signal selects the channel in which the read or write operation occurs.
$\overline{WR}$	35	I	**WRITE:** When the SCC is selected this signal indicates a write operation. The coincidence of $\overline{\text{RD}}$ and $\overline{\text{WR}}$ is interpreted as a reset.
$\overline{RD}$	36	I	**READ:** This signal indicates a read operation and when the SCC is selected, enables the SCC's bus drivers. During the Interrupt Acknowledge cycle, this signal gates the interrupt vector onto the bus if the SCC is the highest priority device requesting an interrupt.

GENERAL DESCRIPTION

The Intel 82350 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. The SCC functions as a serial-to-parallel, parallel-to-serial convertor/controller. The SCC can be software-configured to satisfy a wide range of serial communications applications. The device contains sophisticated internal functions including on-chip baud rate generators, digital phase locked loops, various data encoding and decoding schemes, and crystal oscillators that reduce the need for external logic.

In addition, diagnostic capabilities—automatic echo and local loopback—allow the user to detect and isolate a failure in the network. They greatly improve the reliability and fault isolation of the system.

The SCC handles Asynchronous formats, Synchronous byte-oriented protocols such as IBM Bisync, and Synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device supports virtually any serial data transfer application (Terminal, Personal Computer, Peripherals, Industrial Controller, Telecommunication sytem, etc.).

The 82530 can generate and check CRC codes in any Synchronous mode and can be programmed to check data integrity in various modes. The SCC also has facilities for modem control in both channels. In applications where these controls are not needed, the modem control can be used for general purpose I/O.

The Intel 82530 is designed to support Intel's MCS51/96, iAPX 86/88 and iAPX 186/188 families.

ARCHITECTURE

The 82530 internal structure includes two full-duplex channels, two baud rate generators, internal control and interrupt logic, and a bus interface to a non-multiplexed CPU bus. Associated with each channel are a number of read and write registers for mode control and status information, as well as logic necessary to interface modems or other external devices.

The logic for both channels provides formats, synchronization, and validation for data transferred to and from the channel interface. The modem control inputs are monitored by the control logic under program control. All of the modem control signals are general-purpose in nature and can optionally be used for functions other than modem control.

The register set for each channel includes ten control (write) registers, two synchronous character (write) registers, and four status (read) registers. In addition, each baud rate generator has two (read/write) registers for holding the time constant that determines the baud rate. Finally, associated with the interrupt logic is a write register for the interrupt vector accessible through either channel, a write-only Master Interrupt Control register and three read registers; one containing the vector with status information (Channel B only), one containing the vector without status (A only), and one containing the Interrupt Pending bits (A only).

The registers for each channel are designated as follows:

WR0–WR15—Write Registers 0 through 15.

RR0–RR3, RR10, RR12, RR13, RR15—Read Registers 0 through 3, 10, 12, 13, 15.

Table 2 lists the functions assigned to each read or write register. The SCC contains only one WR2 and WR9, but they can be accessed by either channel. All other registers are paired (one for each channel).

DATA PATH

The transmit and receive data path illustrated in Figure 3 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement, in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. Incoming data is routed through one of several paths (data or CRC) depending on the selected mode (the character length in asynchronous modes also determines the data path).

The transmitter has an 8-bit transmit data buffer register loaded from the internal data bus and a 20-bit transmit shift register that can be loaded either from the sync-character registers or from the transmit data register. Depending on the operational mode, outgoing data is routed through one of four main paths before it is transmitted from the Transmit Data output (TxD).

Table 2. Read and Write Register Functions

READ REGISTER FUNCTIONS		WRITE REGISTER FUNCTIONS	
RR0	Transmit/Receive buffer status and External status	**WR0**	CRC initialize, initialization commands for the various modes, shift right/shift left command
RR1	Special Receive Condition status	**WR1**	Transmit/Receive interrupt and data transfer mode definition
RR2	Modified interrupt vector (Channel B only) Unmodified interrupt (Channel A only)	**WR2**	Interrupt vector (accessed through either channel)
RR3	Interrupt Pending bits (Channel A only)	**WR3**	Receive parameters and control
RR8	Receive buffer	**WR4**	Transmit/Receive miscellaneous parameters and modes
RR10	Miscellaneous status	**WR5**	Transmit parameters and controls
RR12	Lower byte of baud rate generator time constant	**WR6**	Sync characters or SDLC address field
RR13	Upper byte of baud rate generator time constant	**WR7**	Sync character or SDLC flag
RR15	External/Status interrupt information	**WR8**	Transmit buffer
		WR9	Master interrupt control and reset (accessed through either channel)
		WR10	Miscellaneous transmitter/receiver control bits
		WR11	Clock Mode control
		WR12	Lower Byte of baud rate generator time constant
		WR13	Upper Byte of baud rate generator time constant
		WR14	Miscellaneous control bits
		WR15	External/Status interrupt control

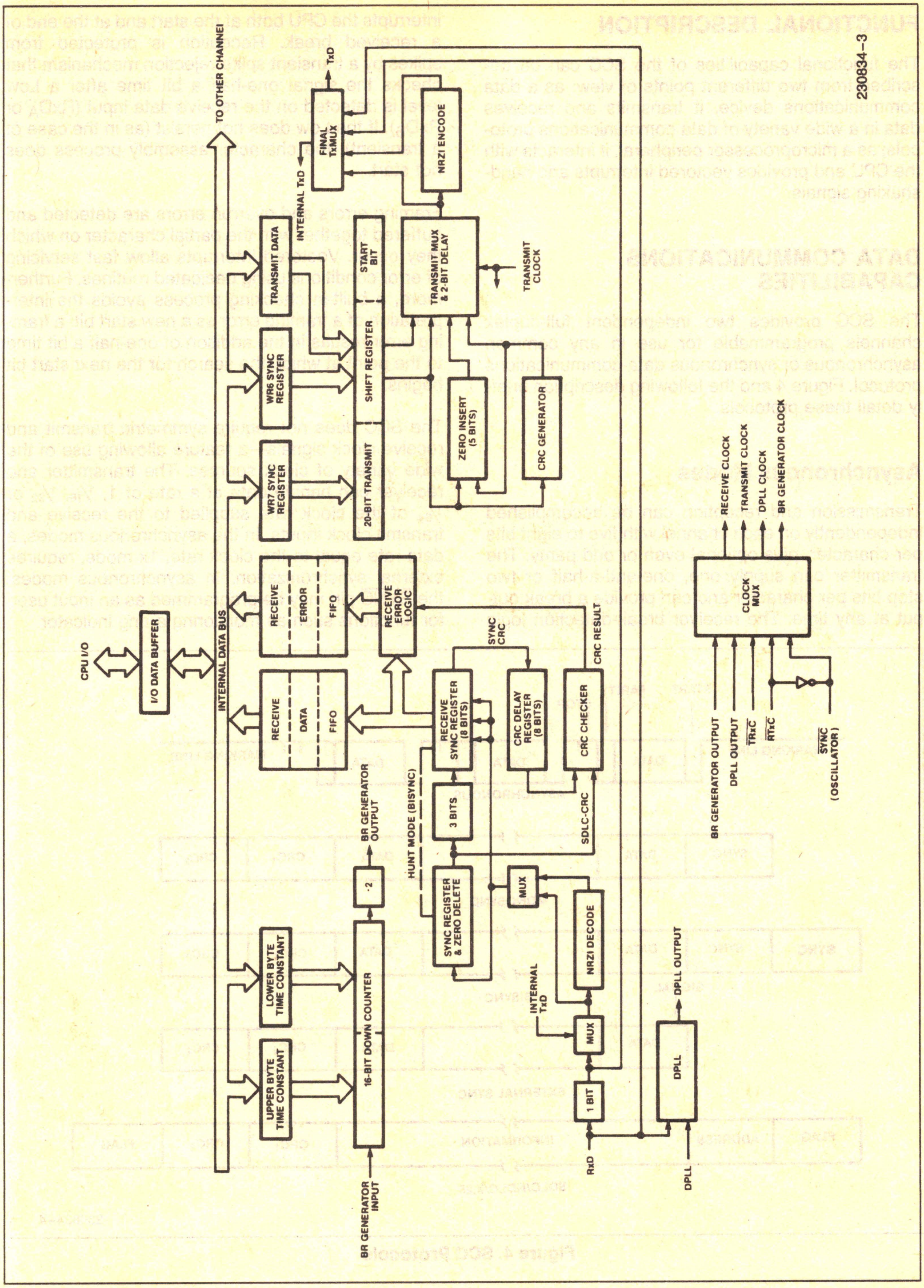


Figure 3. Data Path

FUNCTIONAL DESCRIPTION

The functional capabilities of the SCC can be described from two different points of view: as a data communications device, it transmits and receives data in a wide variety of data communications protocols; as a microprocessor peripheral, it interacts with the CPU and provides vectored interrupts and handshaking signals.

DATA COMMUNICATIONS CAPABILITIES

The SCC provides two independent full-duplex channels programmable for use in any common asynchronous or synchronous data-communications protocol. Figure 4 and the following description briefly detail these protocols.

Asynchronous Modes

Transmission and reception can be accomplished independently on each channel with five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one-and-a-half or two stop bits per character and can provide a break output at any time. The receiver break-detection logic interrupts the CPU both at the start and at the end of a received break. Reception is protected from spikes by a transient spike-rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the receive data input (RxD_A or RxD_B). If the Low does not persist (as in the case of a transient), the character assembly process does not start.

Framing errors and overrun errors are detected and buffered together with the partial character on which they occur. Vectored interrupts allow fast servicing of error conditions using dedicated routines. Furthermore, a built-in checking process avoids the interpretation of a framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit begins.

The SCC does not require symmetric transmit and receive clock signals—a feature allowing use of the wide variety of clock sources. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32 or 1/64 of the clock rate supplied to the receive and transmit clock inputs. In the asynchronous modes, a data rate equal to the clock rate, 1x mode, requires external synchronization. In asynchronous modes, the $\overline{\text{SYNC}}$ pin may be programmed as an input used for functions such as monitoring a ring indicator.

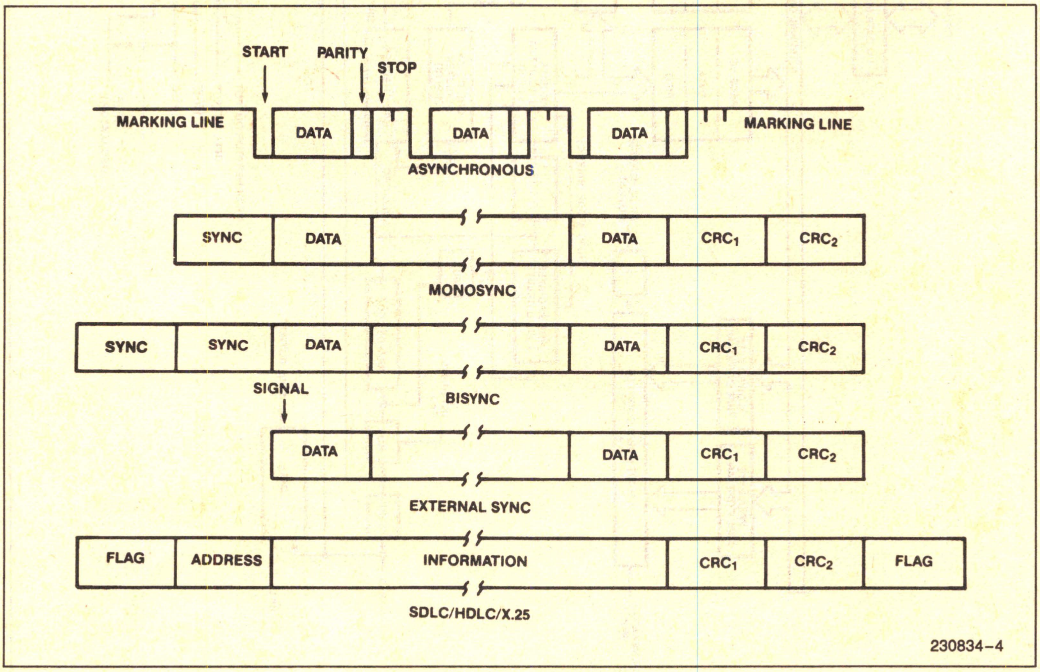


Figure 4. SCC Protocols

Synchronous Modes

The SCC supports both byte-oriented and bit-oriented synchronous communication. Synchronous-byte-oriented protocols can be handled in several modes allowing character synchronization with a 6-bit or 8-bit synchronous character (Monosync), any 12-bit or 16-bit synchronous pattern (Bisync), or with an external synchronous signal. Leading synchronous characters can be removed without interrupting the CPU.

5- or 7-bit synchronous characters are detected with 8- or 16-bit patterns in the SCC by overlapping the larger pattern across multiple incoming synchronous characters as shown in Figure 5.

CRC checking for Synchronous byte-oriented mode is delayed by one character time so that the CPU may disable CRC checking on specific characters. This permits the implementation of protocols such as IBM Bisync.

Both CRC-16 ($X^{16} + X^{15} + X^2 + 1$) and CCITT ($X^{16} + X^{12} + X^5 + 1$) error checking polynomials are supported. Either polynomial may be selected in all synchronous modes. Users may preset the CRC generator and checker to all 1s or all 0s. The SCC also provides a feature that automatically transmits CRC data when no other data is available for transmission. This allows for high-speed transmissions under DMA control, with no need for CPU intervention at the end of a message. When there is no data or CRC to send in synchronous modes, the transmitter inserts 6-, 8-, or 16-bit synchronous characters, regardless of the programmed character length.

The SCC supports synchronous bit-oriented protocols, such as SDLC and HDLC, by performing automatic flag sending, zero insertion, and CRC generation. A special command can be used to abort a frame in transmission. At the end of a message, the SCC automatically transmits the CRC and trailing flag when the transmitter underruns. The transmitter may also be programmed to send an idle line consisting of continuous flag characters or a steady marking condition.

If a transmit underrun occurs in the middle of a message, an external status interrupt warns the CPU of this status change so that an abort may be issued. The SCC may also be programmed to send an abort itself in case of an underrun, relieving the CPU of this task. One to eight bits per character can be sent allowing reception of a message with no prior information about the character structure in the information field of a frame.

The receiver automatically acquires synchronization on the leading flag of a frame in SDLC or HDLC mode and provides a synchronization signal on the $\overline{\text{SYNC}}$ pin (an interrupt can also be programmed). The receiver can be programmed to search for frames addressed by a single byte (or four bits within a byte) of a user-selected address or to a global broadcast address. In this mode, frames not matching either the user-selected or broadcast address are ignored. The number of address bytes can be extended under software control. For receiving data, an interrupt on the first received character, or an interrupt on every character, or on special condition only (end-of-frame) can be selected. The receiver automatically deletes all 0s inserted by the transmitter during character assembly. CRC is also calculated and is automatically checked to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers. In SDLC mode, the SCC must be programmed to use the SDLC CRC polynomial, but the generator and checker may be preset to all 1s or all 0s. The CRC is inverted before transmission and the receiver checks against the bit pattern 0001110100001111.

NRZ, NRZI or FM coding may be used in any 1X mode. The parity options available in asynchronous modes are available in synchronous modes.

The SCC can be conveniently used under DMA control to provide high-speed reception or transmission.

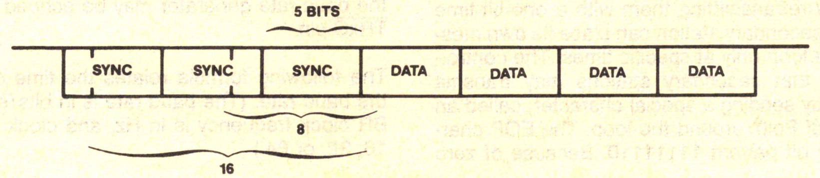


Figure 5. Detecting 5- or 7-Bit Synchronous Characters

In reception, for example, the SCC can interrupt the CPU when the first character of a message is received. The CPU then enables the DMA to transfer the message to memory. The SCC then issues an end-of-frame interrupt and the CPU can check the status of the received message. Thus, the CPU is freed for other service while the message is being received. The CPU may also enable the DMA first and have the SCC interrupt only on end-of-frame. This procedure allows all data to be transferred via DMA.

SDLC LOOP MODE

The SCC supports SDLC Loop mode in addition to normal SDLC. In a loop topology, there is a primary controller station that manages the message traffic flow and any number of secondary stations. In Loop mode, the SCC performs the functions of a secondary station while an SCC operating in regular SDLC mode can act as a controller (Figure 6).

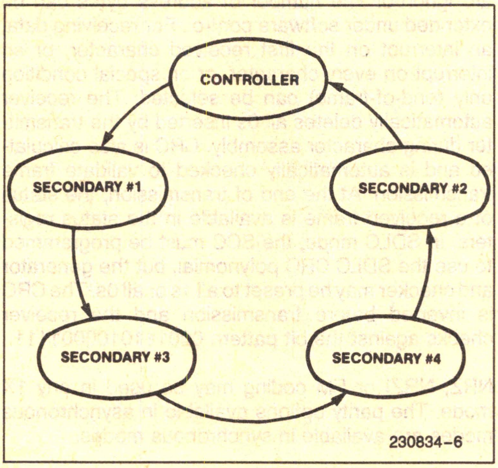


Figure 6. An SDLC Loop

A secondary station in an SDLC Loop is always listening to the messages being sent around the loop, and in fact must pass these messages to the rest of the loop by retransmitting them with a one-bit-time delay. The secondary station can place its own message on the loop only at specific times. The controller signals that secondary stations may transmit messages by sending a special character, called an EOP (End of Poll), around the loop. The EOP character is the bit pattern 11111110. Because of zero insertion during messages, this bit pattern is unique and easily recognized.

When a secondary station has a message to transmit and recognizes an EOP on the line, it changes the last binary one of the EOP to a zero before transmission. This has the effect of turning the EOP into a flag sequence. The secondary station now places its message on the loop and terminates the message with an EOP. Any secondary stations further down the loop with messages to transmit can then append their messages to the message of the first secondary station by the same process. Any secondary stations without messages to send merely echo the incoming messages and are prohibited from placing messages on the loop (except upon recognizing an EOP).

SDLC Loop mode is a programmable option in the SCC. NRZ, NRZI, and FM coding may all be used in SDLC Loop mode.

BAUD RATE GENERATORS

Each channel in the SCC contains a programmable Baud rate generator. Each generator consists of two 8-bit time constant registers that form a 16-bit time constant, a 16-bit down counter, and a flip-flop on the output producing a square wave. On startup, the flip-flop on the output is set in a High state, the value in the time constant register is loaded into the counter, and the counter start counting down. The output of the baud rate generator toggles upon reaching zero, the value in the time constant register is loaded into the counter, and the process is repeated. The time constant may be changed at any time, but the new value does not take effect until the next load of the counter.

The output of the baud rate generator may be used as either the transmit clock, the receive clock, or both. It can also drive the digital phase-locked loop (see next section).

If the receive clock or transmit clock is not programmed to come from the $\overline{TRxC}$ pin, the output of the baud rate generator may be echoed out via the $\overline{TRxC}$ pin.

The following formula relates the time constant to the baud rate. (The baud rate is in bits/second, the BR clock frequency is in Hz, and clock mode is 1, 16, 32, or 64.)

$$\text{time constant} = \frac{\text{BR clock frequency}}{2 \times \text{baud rate} \times \text{clock mode}} - 2$$

Table 3. Time Constant Values for Standard Baud Rates at BR Clock = 3.9936 MHz

Rate (BAUD)	Time Constant (decimal notation)	Error
19200	102	—
9600	206	—
7200	275	0.12%
4800	414	—
3600	553	0.06%
2400	830	—
2000	996	0.04%
1800	1107	0.03%
1200	1662	—
600	3326	—
300	6654	—
150	13310	—
134.5	14844	0.0007%
110	18151	0.0015%
75	26622	—
50	39934	—

DIGITAL PHASE LOCKED LOOP

The SCC contains a digital phase locked-loop (DPLL) to recover clock information from a datastream with NRZI or FM encoding. The DPLL is driven by a clock that is nominally 32 (NRZI) or 16 (FM) times the data rate. The DPLL uses this clock, along with the datastream, to construct a clock for the data. This clock may then be used as the SCC receive clock, the transmit clock, or both.

For NRZI coding, the DPLL counts the 32X clock to create nominal bit times. As the 32X clock is counted, the DPLL is searching the incoming datastream for edges (either 1/0 or 0/1). Whenever an edge is detected, the DPLL makes a count adjustment (during the next counting cycle), producing a terminal count closer to the center of the bit cell.

For FM encoding, the DPLL still counts from 1 to 31, but with a cycle corresponding to two bit times. When the DPLL is locked, the clock edges in the datastream should occur between counts 15 and 16 and between counts 31 and 0. The DPLL looks for edges only during a time centered on the $^{15}/_{16}$ counting transition.

The 32X clock for the DPLL can be programmed to come from either the $\overline{\text{RTxC}}$ input or the output of the baud rate generator. The DPLL output may be programmed to be echoed out of the SCC via the $\overline{\text{TRxC}}$ pin (if this pin is not being used as an input).

DATA ENCODING

The SCC may be programmed to encode and decode the serial data in four different ways (Figure 7). In NRZ encoding, a 1 is represented by a High level and a 0 is represented by a Low level. In NRZI encoding, as 1 is represented by no change in level and a 0 is represented by a change in level. In FM_1 (more properly, bi-phase mark) a transition occurs at the beginning of every bit cell. A 1 is represented by an additional transition at the center of the bit cell and a 0 is represented by no additional transition at the center of the bit cell. In FM_0 (bi-phase space), a transition occurs at the beginning of every bit cell. A 0 is represented by an additional transition at the center of the bit cell, and a 1 is represented by no additional transition at the center of the bit cell. In addition to these four methods, the SCC can be used to decode Manchester (bi-phase level) data by using the DPLL in the FM mode and programming the receiver for NRZ data. Manchester encoding always produces a transition at the center of the bit cell. If the transition is 0/1 the bit is a 0. If the transition is 1/0 the bit is a 1.

AUTO ECHO AND LOCAL LOOPBACK

The SCC is capable of automatically echoing everything it receives. This feature is useful mainly in asynchronous modes, but works in synchronous and SDLC modes as well. In Auto Echo mode TxD is RxD. Auto Echo mode can be used with NRZI or FM encoding with no additional delay, because the datastream is not decoded before retransmission. In Auto Echo mode, the $\overline{\text{CTS}}$ input is ignored as a transmitter enable (although transitions on this input can still cause interrupts if programmed to do so). In this mode, the transmitter is actually bypassed and the programmer is responsible for disabling transmitter interrupts and $\overline{\text{READY/REQUEST}}$ on transmit.

The SCC is also capable of local loopback. In this mode, TxD is RxD just as in Auto Echo mode. However, in Local Loopback mode, the internal transmit data is tied to the internal receive data and RxD is ignored (except to be echoed out via TxD). $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ inputs are also ignored as transmit and receive enables. However, transitions on these inputs can still cause interrupts. Local Loopback works in asynchronous, synchronous and SDLC modes with NRZ, NRZI or FM coding of the data stream.

SERIAL BIT RATE

To run the 82530 (4 MHz/6 MHz) at 1/1.5 Mbps the receive and transmit clocks must be externally generated and synchronized to the system clock. If the serial clocks ($\overline{\text{RTxC}}$ and $\overline{\text{TRxC}}$) and the system clock (CLK) are asynchronous, the maximum bit rate is 880 Kbps/1.3 Mbps. For self-clocked operation, i.e using the on chip DPLL, the maximum bit rate is 125/187 Kbps if NRZI coding is used and 250/375 Kbps if FM coding is used.

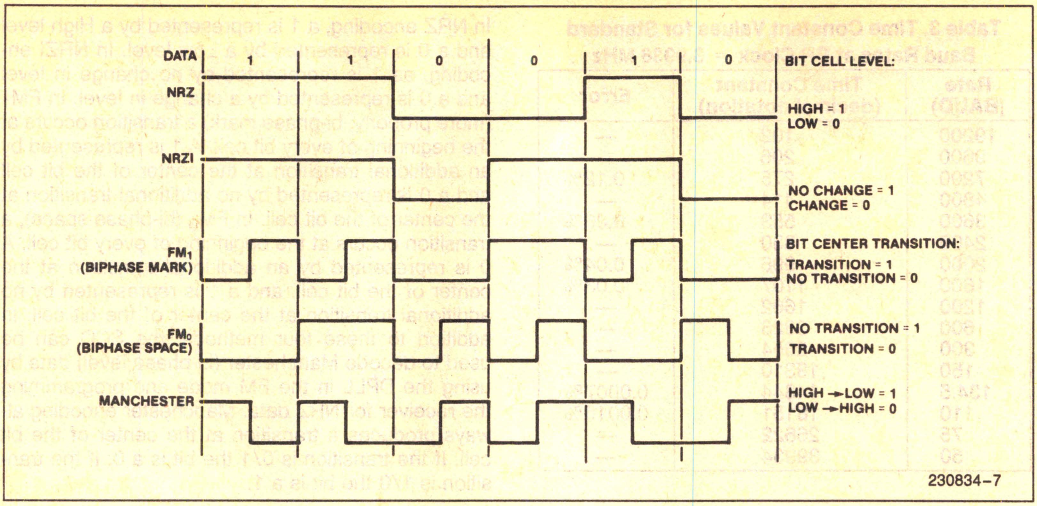


Figure 7. Data Encoding Methods

Table 4. Maximum Bit Rates

Mode	System Clock	System Clock/ Serial Clock	Serial Bit Rate	Conditions
Serial clocks generated externally	4 MHz	4	1 Mbps	Serial clocks synchronized with system clock. Refer to parameter #3 and 10 in general timings.
	6 MHz	4	1.5 Mbps	Serial clocks synchronized with system clock. Refer to parameter #3 and #10 in general timings.
	4 MHz	4.5	880 Kbps	Serial clocks and system clock asynchronous.
	6 MHz	4.5	1.3 Mbps	Serial clocks and system clock asynchronous
Self-clocked operation				
NRZI	4 MHz	32	125 Kbps	
	6 MHz	32	187 Kbps	
FM	4 MHz	16	250 Kbps	
	6 MHz	16	375 Kbps	
ASYNC	4 MHz	16	62.5 Kbps	
	6 MHz	16	93.75 Kbps	

I/O INTERFACE CAPABILITIES

The SCC offers the choice of Polling, Interrupt (vectored or nonvectored) and Block Transfer modes to transfer data, status, and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

POLLING

All interrupts are disabled. Three status registers in the SCC are automatically updated whenever any function is performed. For example, end-of-frame in SDLC mode sets a bit in one of these status registers. The idea behind polling is for the CPU to periodically read a status register until the register contents indicate the need for data to be transferred. Only one register needs to be read; depending on its contents, the CPU either writes data, reads data, or continues. Two bits in the register indicate the need for data transfer. An alternative is a poll of the Interrupt Pending register to determine the source of an interrupt. The status for both channels resides in one register.

INTERRUPTS

When a SCC responds to an Interrupt Acknowledge signal ($\overline{\text{INTA}}$) from the CPU, an interrupt vector may be placed on the data bus. This vector is written in WR2 and may be read in RR2A or RR2B (Figures 9 and 10).

To speed interrupt response time, the SCC can modify three bits in this vector to indicate status. If the vector is read in Channel A, status is never included; if it is read in Channel B, status is always included.

Each of the six sources of interrupts in the SCC (Transmit, Receive and External/Status interrupts in both channels) has three bits associated with the interrupt source: Interrupt Pending (IP), Interrupt Under Service (IUS), and Interrupt Enable (IE). Operation of the IE bits is straightforward. If the IE bit is set for a given interrupt source, then that source can request interrupts. The exception is when the MIE (Master Interrupt Enable) bit in WR9 is reset and no interrupts may be requested. The IE bits are write-only.

The other two bits are related to the interrupt priority chain (Figure 8). As a peripheral, the SCC may request an interrupt only when no higher-priority device is requesting one, e.g., when IEI is High. If the device in question requests an interrupt, it pulls down $\overline{\text{INT}}$. The CPU then responds with $\overline{\text{INTA}}$, and the interrupting device places the vector on the data bus.

In the SCC, the IP bit signals a need for interrupt servicing. When an IP bit is 1 and the IEI input is High, the $\overline{\text{INT}}$ output is pulled Low, requesting an interrupt. In the SCC, if the IE bit is not set by enabling interrupts, then the IP for that source can never be set. The IP bits are readable in RR3A.

The IUS bits signal that an interrupt request is being serviced. If an IUS is set, all interrupt sources of lower priority in the SCC and external to the SCC are prevented from requesting interrupts. The internal interrupt sources are inhibited by the state of the internal daisy chain, while lower priority devices are inhibited by the IEO output of the SCC being pulled Low and propagated to subsequent peripherals. An IUS bit is set during an Interrupt Acknowledge cycle if there are no higher priority devices requesting interrupts.

There are three types of interrupts: Transmit, Receive and External/Status interrupts. Each interrupt type is enabled under program control with Channel A having higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted when the transmit buffer becomes empty. (This implies that the transmitter must have had a data character written into it so that it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on First Receive Character or Special Receive condition.
- Interrupt on all Receive Characters or Special Receive condition.
- Interrupt on Special Receive condition only.

Interrupt-on-First-Character or Special-Condition and Interrupt-on-Special-Condition-Only are typically used with the Block Transfer mode. A Special-Receive-Condition is one of the following: receiver overrun, framing error in Asynchronous mode, End-of-Frame in SDLC mode and, optionally, a parity error. The Special-Receive-Condition interrupt is different from an ordinary receive character available interrupt only in the status placed in the vector

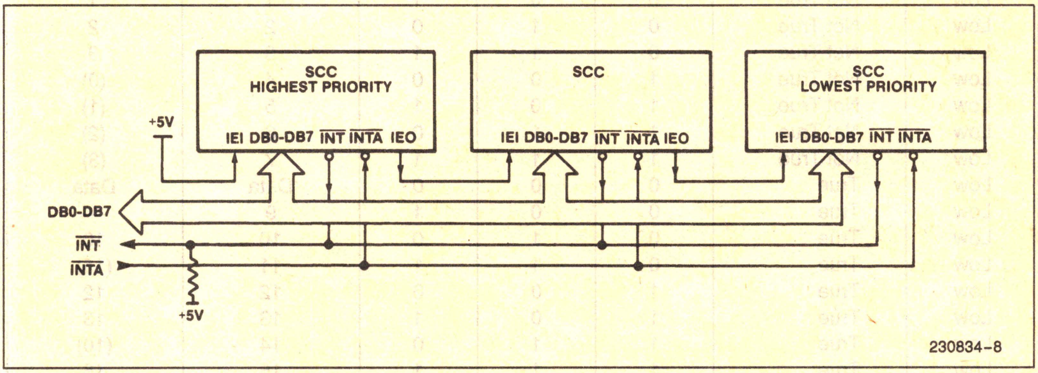


Figure 8. Daisy Chaining SCC's

during the Interrupt-Acknowledge cycle. In Interrupt on First Receive Character, an interrupt can occur from Special Receive conditions any time after the first receive character interrupt.

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{CTS}$, $\overline{CD}$, and $\overline{SYNC}$ pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition, or a zero count in the baud rate generator, or by the detection of a Break (asynchronous mode), Abort (SDLC mode) or EOP (SDLC Loop mode) sequence in the data stream. The interrupt caused by the Abort or EOP has a special feature allowing the SCC to interrupt when the Abort or EOP sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Abort condition in external logic in SDLC mode. In SDLC Loop mode this feature allows secondary stations to recognize the wishes of the primary station to regain control of the loop during a poll sequence.

CPU/DMA BLOCK TRANSFER

The SCC provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers. The Block Transfer mode uses the $\overline{READY}/\overline{REQUEST}$ output in conjunction with the $\overline{READY}/\overline{REQUEST}$ bits in WR1. The $\overline{READY}/\overline{REQUEST}$ output can be defined under software control as a $\overline{READY}$ line in the CPU Block Transfer mode (WR1; D6 = 0) or as a request line in the DMA Block Transfer mode (WR1; D6 = 1). To a DMA controller, the SCC $\overline{REQUEST}$ output indicates that the SCC is ready to transfer data to or from memory. To the CPU, The $\overline{READY}$ line indicates that the SCC is not ready to transfer data, thereby requesting that the CPU extend the I/O cycle. The $\overline{DTR}/\overline{REQUEST}$ line allows full-duplex operation under DMA control.

PROGRAMMING

Each channel has fifteen Write registers that are individually programmed from the system bus to configure the functional personality of each channel. Each channel also has eight Read registers from which the system can read Status, Baud rate, or Interrupt information.

Only the four data registers (Read, Write for channels A and B) are directly selected by a High on the D/$\overline{C}$ input and the appropriate levels on the $\overline{RD}$, $\overline{WR}$ and A/$\overline{B}$ pins. All other registers are addressed indirectly by the content of Write Register 0 in conjunction with a Low on the D/$\overline{C}$ input and the appropriate levels on the $\overline{RD}$, $\overline{WR}$ and A/$\overline{B}$ pins. If bit 4 in WW0 is 1 and bits 5 and 6 are 0 then bits 0, 1, 2 address the higher registers 8 through 15. If bits 4, 5, 6 contain a different code, bits 0, 1, 2 address the lower registers 0 through 7 as shown on Table 5.

Writing to or reading from any register except RR0, WR0 and the Data Registers thus involves two operations.

Table 5. Register Addressing

D/C	"Point High" Code in WR0	D2 (in WR0)	D1 (in WR0)	D0	Write Register	Read Register
High	Either Way	X	X	X	Data	Data
Low	Not True	0	0	0	0	0
Low	Not True	0	0	1	1	1
Low	Not True	0	1	0	2	2
Low	Not True	0	1	1	3	3
Low	Not True	1	0	0	4	(0)
Low	Not True	1	0	1	5	(1)
Low	Not True	1	1	0	6	(2)
Low	Not True	1	1	1	7	(3)
Low	True	0	0	0	Data	Data
Low	True	0	0	1	9	—
Low	True	0	1	0	10	10
Low	True	0	1	1	11	(15)
Low	True	1	0	0	12	12
Low	True	1	0	1	13	13
Low	True	1	1	0	14	(10)
Low	True	1	1	1	15	15

First write the appropriate code into WR0, then follow this by a write or read operation on the register thus specified. Bits 0 through 4 in WW0 are automatically cleared after this operation, so that WW0 then points to WR0 or RR0 again.

Channel A/Channel B selection is made by the A/$\overline{B}$ input (High = A, Low = B)

The system program first issues a series of commands to initialize the basic mode of operation. This is followed by other commands to qualify conditions within the selected mode. For example, the Asynchronous mode, character length, clock rate, number of stop bits, even or odd parity might be set first. Then the interrupt mode would be set, and finally, receiver or transmitter enable.

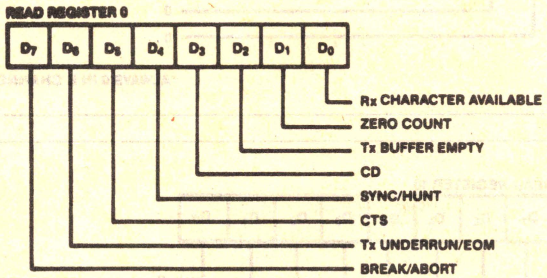
READ REGISTERS

The SCC contains eight read registers (actually nine, counting the receive buffer (RR8) in each channel). Four of these may be read to obtain status information (RR0, RR1, RR10, and RR15). Two registers (RR12 and RR13) may be read to determine the baud rate generator time constant. RR2 contains either the unmodified interrupt vector (Channel A) or the vector modified by status information (Channel B). RR3 contains the Interrupt Pending (IP) bits (Channel A). Figure 9 shows the formats for each read register.

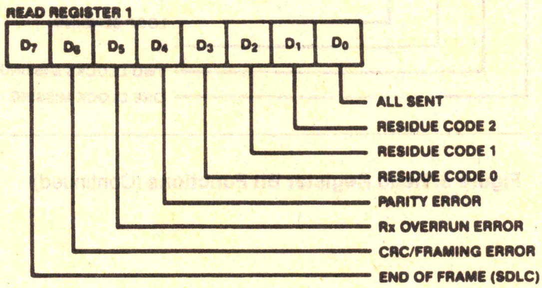
The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring: e.g. when the interrupt vector indicates a Special Receive Condition interrupt, all the appropriate error bits can be read from a single register (RR1).

WRITE REGISTERS

The SCC contains 15 write registers (16 counting WR8, the transmit buffer) in each channel. These write registers are programmed separately to configure the functional "personality" of the channels. In addition, there are two registers (WR2 and WR9) shared by the two channels that may be accessed through either of them. WR2 contains the interrupt vector for both channels, while WR9 contains the interrupt control bits. Figure 10 shows the format of each write register.

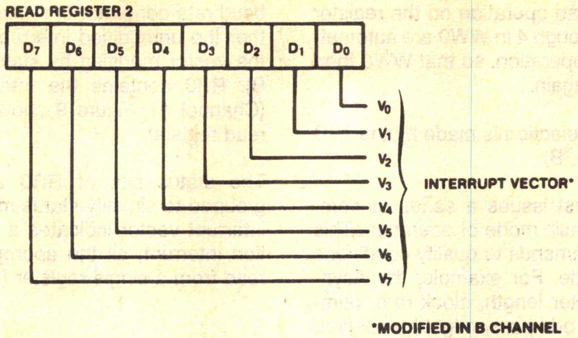


230834-9

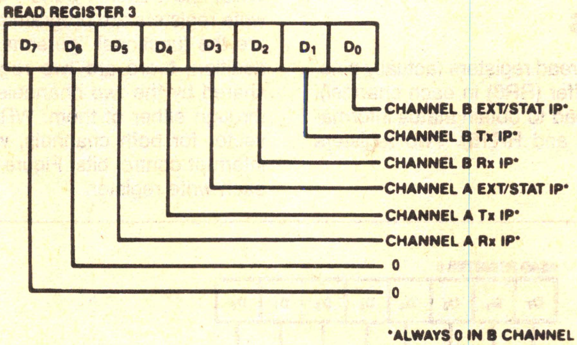


230834-10

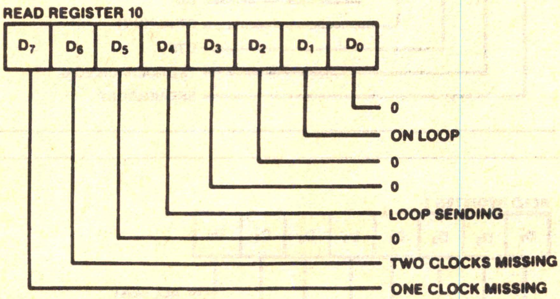
Figure 9. Read Register Bit Functions



230834-11

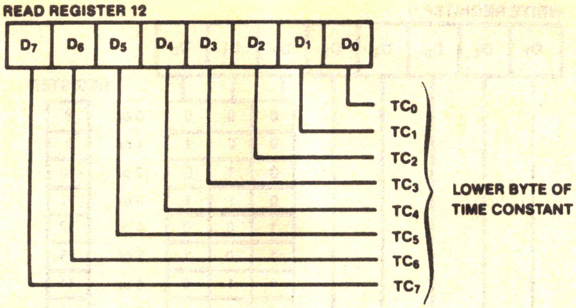


230834-12

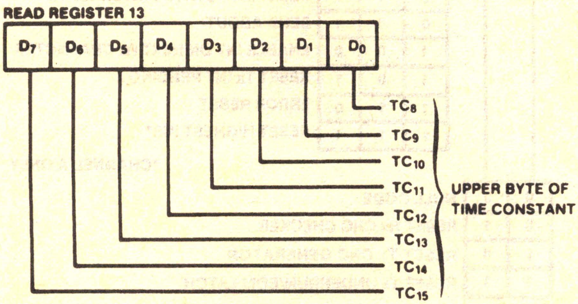


230834-13

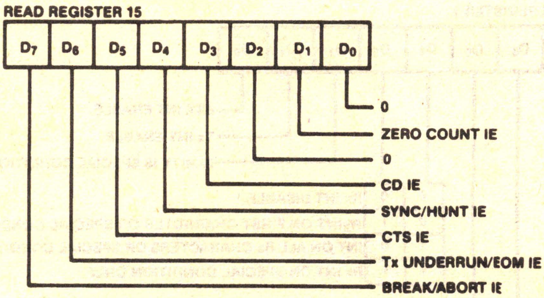
Figure 9. Read Register Bit Functions (Continued)



230834-14

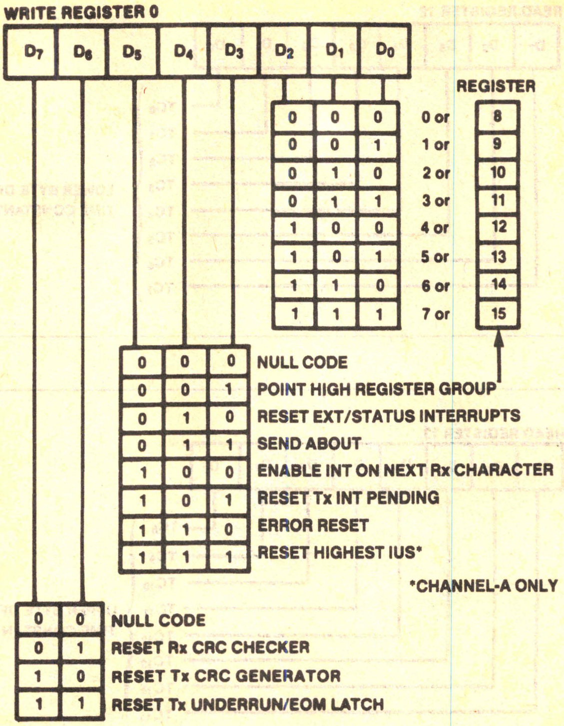


230834-15



230834-16

Figure 9. Read Register Bit Functions (Continued)



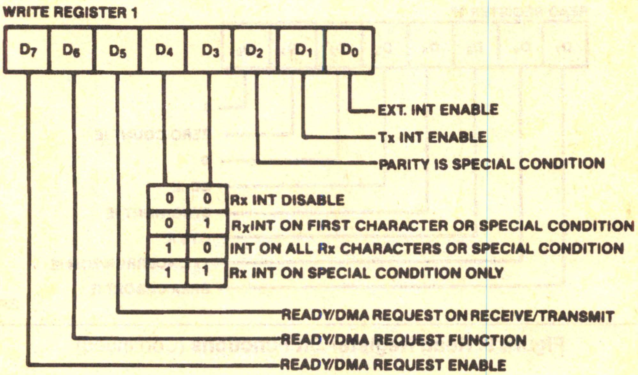
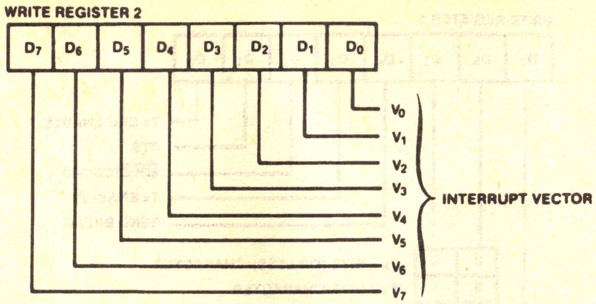
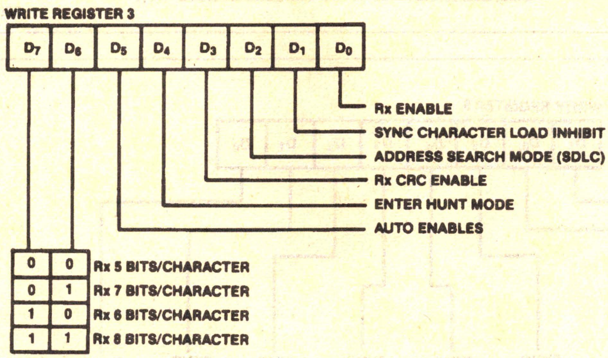


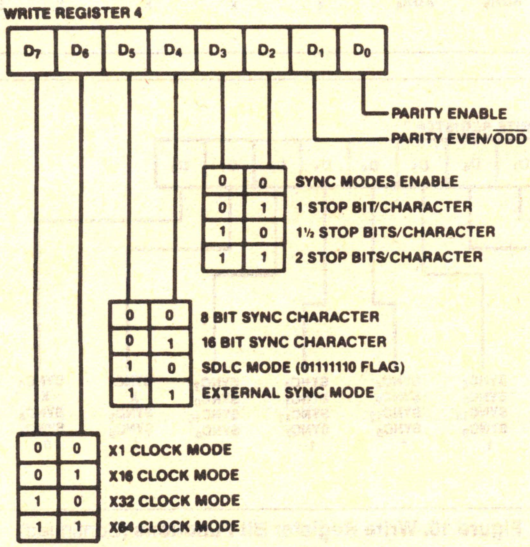
Figure 10. Write Register Bit Functions



230834-21

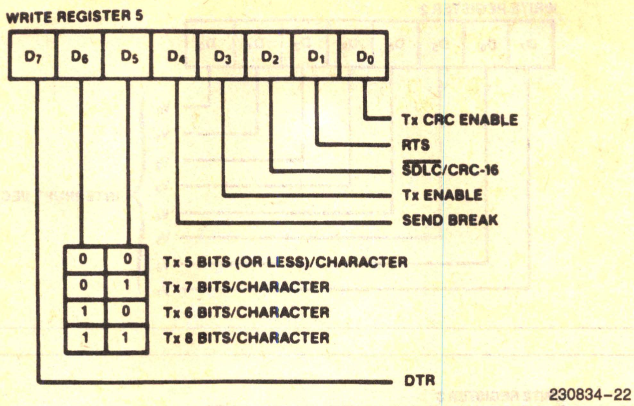


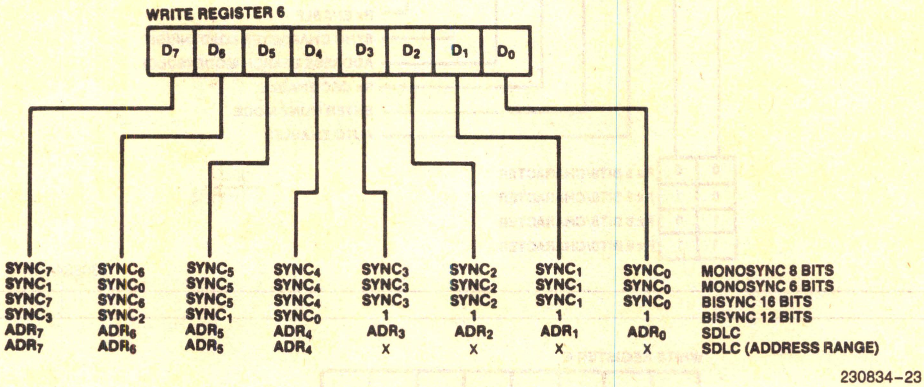
230834-18



230834-20

Figure 10. Write Register Bit Functions (Continued)





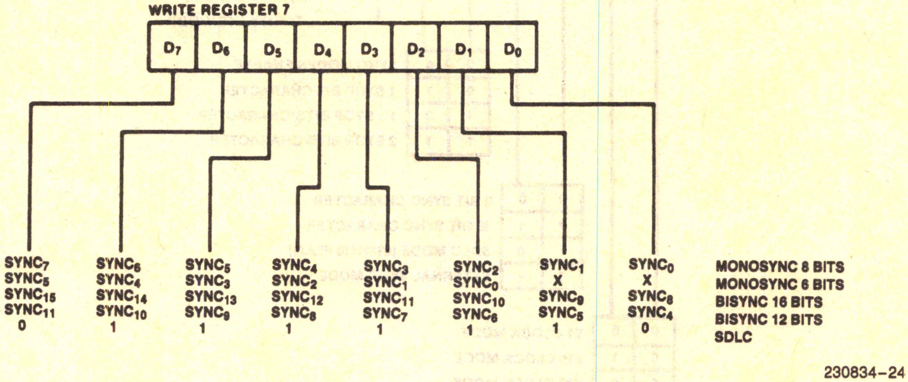
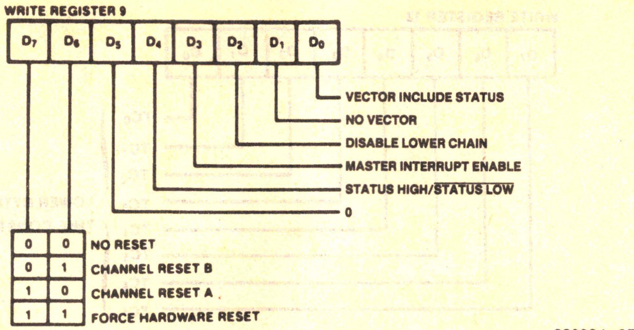
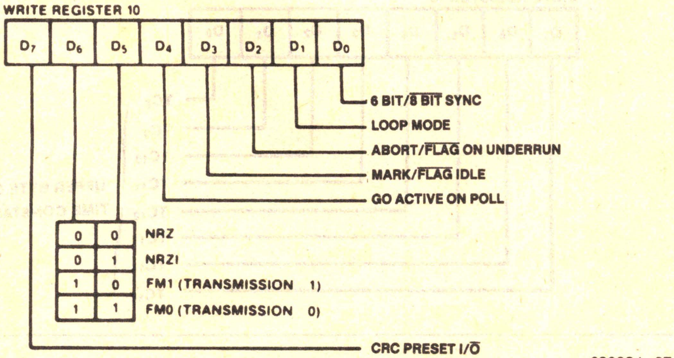


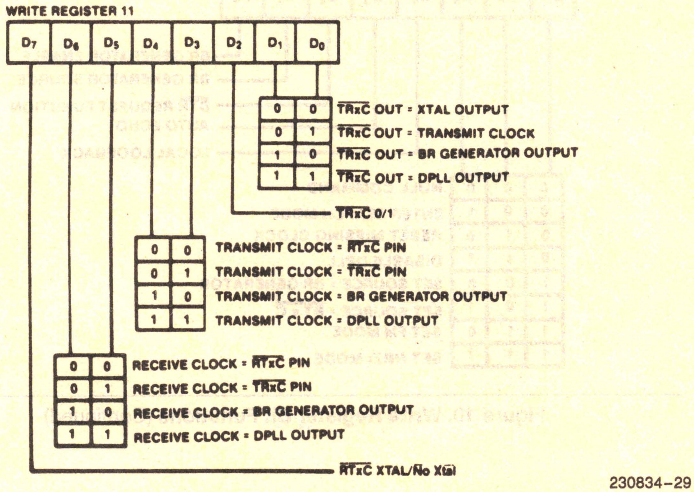
Figure 10. Write Register Bit Functions (Continued)



230834-25

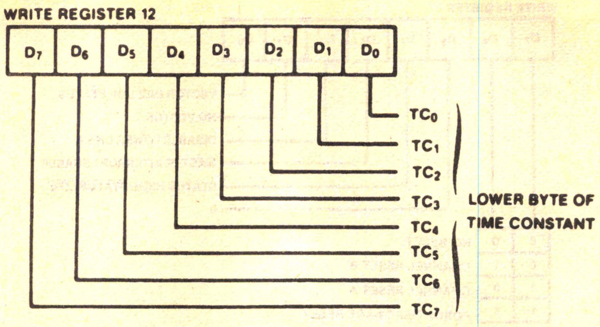


230834-27

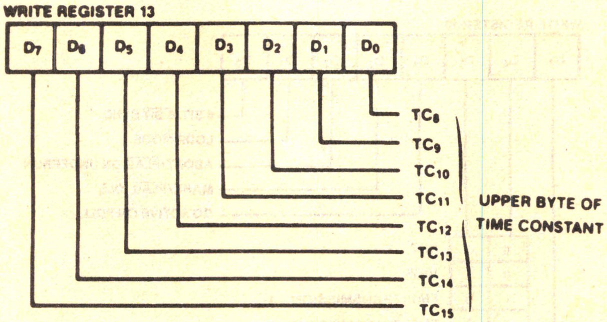


230834-29

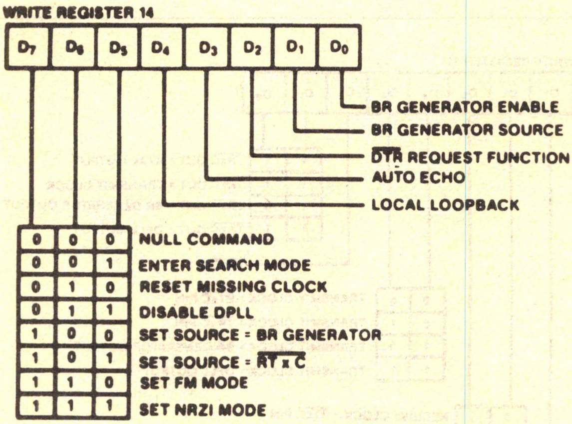
Figure 10. Write Register Bit Functions (Continued)



230834-26



230834-28



230834-30

Figure 10. Write Register Bit Functions (Continued)

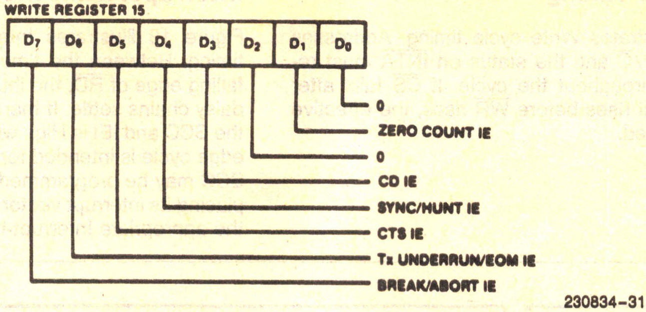


Figure 10. Write Register Bit Functions (Continued)

82530 TIMING

The SCC generates internal control signals from $\overline{WR}$ and $\overline{RD}$ that are related to CLK. Since CLK has no phase relationship with $\overline{WR}$ and $\overline{RD}$, the circuitry generating these internal control signals must provide time for metastable conditions to disappear. This gives rise to a recovery time related to CLK. The recovery time applies only between bus transactions involving the SCC. The recovery time required for proper operation is specified from the rising edge of $\overline{WR}$ or $\overline{RD}$ in the first transaction involving the SCC to the falling edge of $\overline{WR}$ or $\overline{RD}$ in the second transaction involving the SCC. This time, T_{REC} must be at least 6 CLK cycles plus 130 ns, for the 82530-6.

Read Cycle Timing

Figure 11 illustrates Read cycle timing. Addresses on A/$\overline{B}$ and D/$\overline{C}$ and the status on $\overline{INTA}$ must remain stable throughout the cycle. If $\overline{CS}$ falls after $\overline{RD}$ falls or if it rises before $\overline{RD}$ rises, the effective $\overline{RD}$ is shortened.

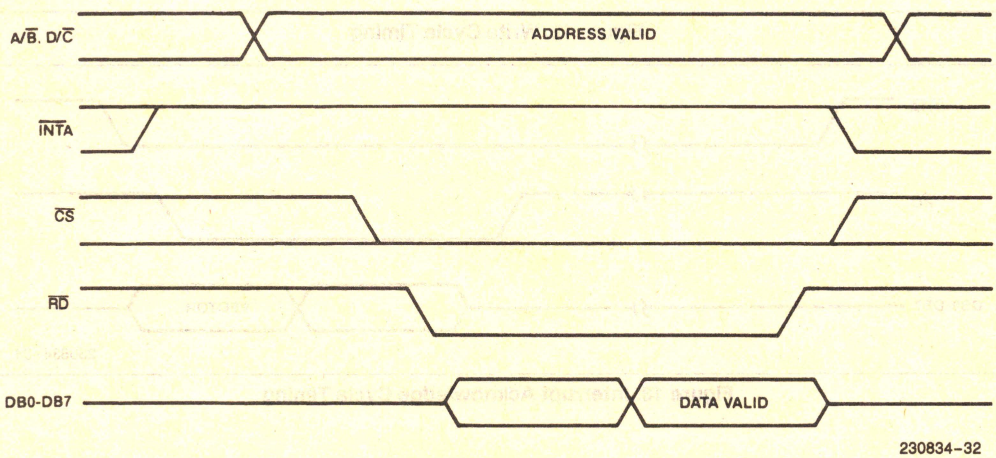


Figure 11. Read Cycle Timing

Write Cycle Timing

Figure 12 illustrates Write cycle timing. Addresses on A/$\overline{B}$ and D/$\overline{C}$ and the status on $\overline{INTA}$ must remain stable throughout the cycle. If $\overline{CS}$ falls after $\overline{WR}$ falls or if it rises before $\overline{WR}$ rises, the effective $\overline{WR}$ is shortened.

Interrupt Acknowledge Cycle Timing

Figure 13 illustrates Interrupt Acknowledge cycle timing. Between the time $\overline{INTA}$ goes Low and the falling edge of $\overline{RD}$, the internal and external IEI/IEO daisy chains settle. If there is an interrupt pending in the SCC and IEI is High when $\overline{RD}$ falls, the Acknowledge cycle is intended for the SCC. In this case, the SCC may be programmed to respond to $\overline{RD}$ Low by placing its interrupt vector on D_0-D_7 and it then sets the appropriate Interrupt-Under-Service internally.

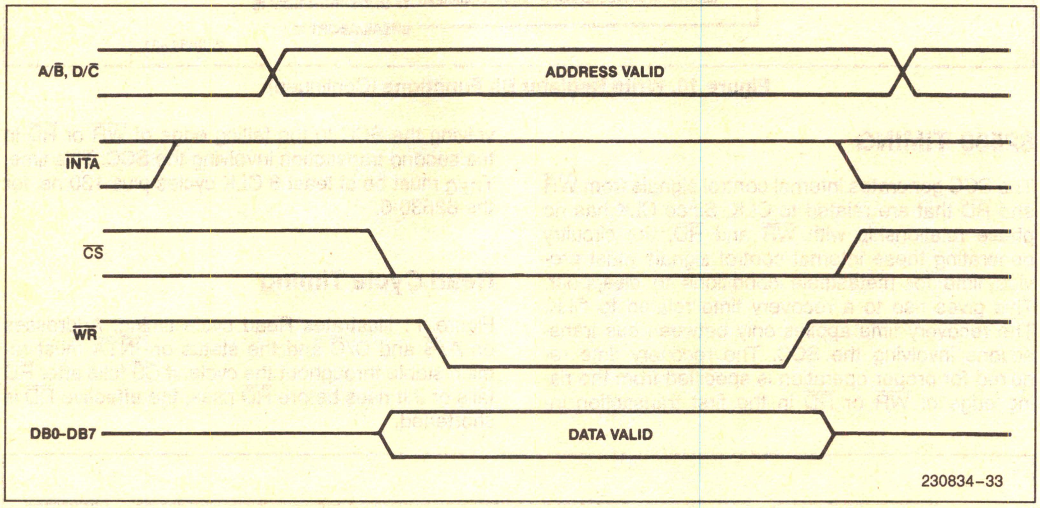


Figure 12. Write Cycle Timing

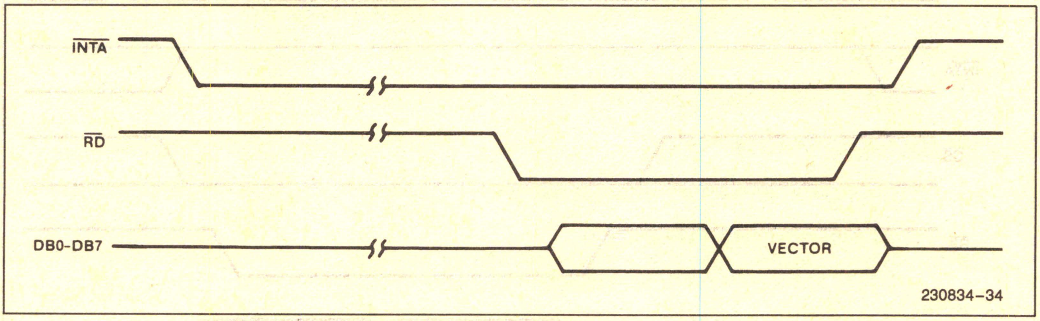


Figure 13. Interrupt Acknowledge Cycle Timing

ABSOLUTE MAXIMUM RATINGS*

Case Temperature
Under Bias .... 0°C to +70°C

Storage Temperature
Ceramic Package .... −65°C to +150°C
Plastic Package .... −40°C to +125°C

Voltage on Any Pin with
Respect to Ground .... −0.5V to +7.0V

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_C = 0°C$ to $70°C$; $V_{CC} = +5V \pm 5\%$

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.4	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		+0.45	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	+2.4		V	$I_{OH} = -250\ \mu A$
I_{IL}	Input Leakage Current		±10	μA	0.4V to 2.4V
I_{OL}	Output Leakage Current		±10	μA	0.4V to 2.4V
I_{CC}	V_{CC} Supply Current		250	mA	

CAPACITANCE $T_C = 25°C$; $V_{CC} = GND = 0V$

Symbol	Parameter	Min	Max	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	$f_c = 1$ MHz
C_{OUT}	Output Capacitance		15	pF	Unmeasured pins returned to GND
$C_{I/O}$	Input/Output Capacitance		20	pF	

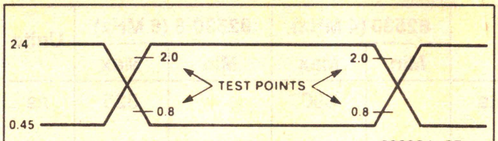
A.C CHARACTERISTICS $T_C = 0°C$ to $+70°C$; $V_{CC} = +5V \pm 5\%$

READ AND WRITE TIMING

Number	Symbol	Parameter	82530 (4 MHz)		82530-6 (6 MHz)		Units
			Min	Max	Min	Max	
1	tCL	CLK Low Time	105	2000	70	1000	ns
2	tCH	CLK High Time	105	2000	70	1000	ns
3	tf	CLK Fall Time		20		10	ns
4	tr	CLK Rise Time		20		15	ns
5	tCY	CLK Cycle Time	250	4000	165	2000	ns
6	tAW	Address to $\overline{WR}\downarrow$ Setup Time	80		0		ns
7	tWA	Address to $\overline{WR}\uparrow$ Hold Time	0		0		ns
8	tAR	Address to $\overline{RD}\downarrow$ Setup Time	80		0		ns
9	tRA	Address to $\overline{RD}\uparrow$ Hold Time	0		0		ns
10	tIC	$\overline{INTA}$ to CLK$\uparrow$ Setup Time	5		5		ns
11	tIW	$\overline{INTA}$ to $\overline{WR}\downarrow$ Setup Time (Note 1)	200		55		ns
12	tWI	$\overline{INTA}$ to $\overline{WR}\uparrow$ Hold Time	0		0		ns
13	tIR	$\overline{INTA}$ to $\overline{RD}\downarrow$ Setup Time (Note 1)	200		55		ns
14	tRI	$\overline{INTA}$ to $\overline{RD}\uparrow$ Hold Time	0		0		ns
15	tCI	$\overline{INTA}$ to CLK$\uparrow$ Hold Time	100		100		ns
16	tCLW	$\overline{CS}$ Low to $\overline{WR}\downarrow$ Setup Time	0		0		ns
17	tWCS	$\overline{CS}$ to $\overline{WR}\uparrow$ Hold Time	0		0		ns
18	tCHW	$\overline{CS}$ High to $\overline{WR}\uparrow$ Setup Time	100		5		ns
19	tCLR	$\overline{CS}$ Low to $\overline{RD}\downarrow$ Setup Time (Note 1)	0		0		ns
20	tRCS	$\overline{CS}$ to $\overline{RD}\uparrow$ Hold Time (Note 1)	0		0		ns
21	tCHR	$\overline{CS}$ High to $\overline{RD}\downarrow$ Setup Time (Note 1)	100		5		ns
22	tRR	$\overline{RD}$ Low Time (Note 1)	390		150		ns
23	Null	Parameter Deleted					
24	tRDI	$\overline{RD}\uparrow$ to Data Not Valid Delay	0		0		ns
25	tRDV	$\overline{RD}\downarrow$ to Data Valid Delay		250		105	ns
26	tDF	$\overline{RD}\uparrow$ to Output Float Delay (Note 2)		70		45	ns

NOTES:
1. Parameter does not apply to Interrupt Acknowledge transactions.
2. Float delay is defined as the time required for a +0.5V change in the output with a maximum D.C. load and minimum A.C. load.

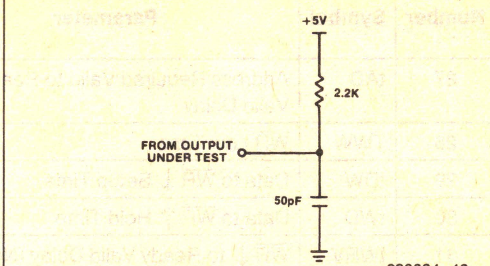
A.C. TESTING INPUT, OUTPUT WAVEFORM



230834-35

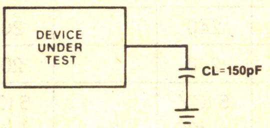
A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

OPEN DRAIN TEST LOAD



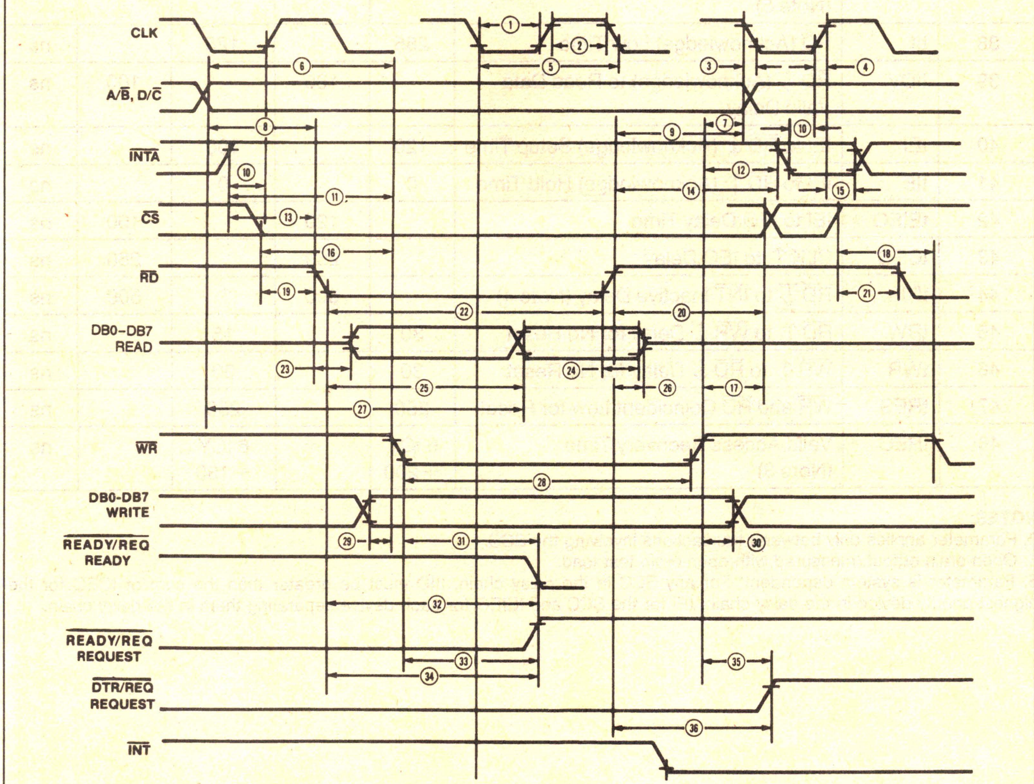
230834-42

A.C. TESTING LOAD CIRCUIT



230834-41

C_L = 150 pF
C_L Includes Jig Capacitance



230834-36

Figure 14. Read and Write Timing

INTERRUPT ACKNOWLEDGE TIMING, RESET TIMING, CYCLE TIMING

Number	Symbol	Parameter	82530 (4 MHz)		82530-6 (6 MHz)		Units
			Min	Max	Min	Max	
27	tAD	Address Required Valid to Read Data Valid Delay		590		325	ns
28	TWW	$\overline{WR}$ Low Time	390		60		ns
29	tDW	Data to $\overline{WR}\downarrow$ Setup Time	0		0		ns
30	tWD	Data to $\overline{WR}\uparrow$ Hold Time	0		0		ns
31	tWRV	$\overline{WR}\downarrow$ to Ready Valid Delay (Note 4)		240		200	ns
32	tRRV	$\overline{RD}\downarrow$ to Ready Valid Delay (Note 4)		240		200	ns
33	tWRI	$\overline{WR}\downarrow$ to $\overline{READY}/\overline{REQ}$ Not Valid Delay		240		200	ns
34	tRRI	$\overline{RD}\downarrow$ to $\overline{READY}/\overline{REQ}$ Not Valid Delay		240		200	ns
35	tDWR	$\overline{WR}\uparrow$ to $\overline{DTR}/\overline{REQ}$ Not Valid Delay		5 tCY +300		5 tCY +250	ns
36	tDRD	$\overline{RD}\uparrow$ to $\overline{DTR}/\overline{REQ}$ Not Valid Delay		5 tCY +300		5 tCY +250	ns
37	tIID	$\overline{INTA}$ to $\overline{RD}\downarrow$ (Acknowledge) Delay (Note 5)	250		250		ns
38	tII	$\overline{RD}$ (Acknowledge) Low Time	285		125		ns
39	tIDV	$\overline{RD}\downarrow$ (Acknowledge) to Read Data Valid Delay		190		100	ns
40	tEI	IEI to $\overline{RD}\downarrow$ (Acknowledge) Setup Time	120		100		ns
41	tIE	IEI to $\overline{RD}\uparrow$ (Acknowledge) Hold Time	0		0		ns
42	tEIEO	IEI to IEO Delay Time		120		100	ns
43	tCEQ	CLK $\uparrow$ to IEO Delay		250		250	ns
44	tRII	$\overline{RD}\downarrow$ to $\overline{INT}$ Inactive Delay (Note 4)		500		500	ns
45	tRW	$\overline{RD}\uparrow$ to $\overline{WR}\downarrow$ Delay for No Reset	30		15		ns
46	tWR	$\overline{WR}\uparrow$ to $\overline{RD}\downarrow$ Delay for No Reset	30		30		ns
47	tRES	$\overline{WR}$ and $\overline{RD}$ Coincident Low for Reset	250		250		ns
48	tREC	Valid Access Recovery Time (Note 3)	6 tCY +200		6 tCY +130		ns

NOTES:
3. Parameter applies only between transactions involving the SCC.
4. Open-drain output, measured with open-drain test load.
5. Parameter is system dependent. For any SCC in the daisy chain, tIID must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for the SCC and tEIEO for each device separating them in the daisy chain.

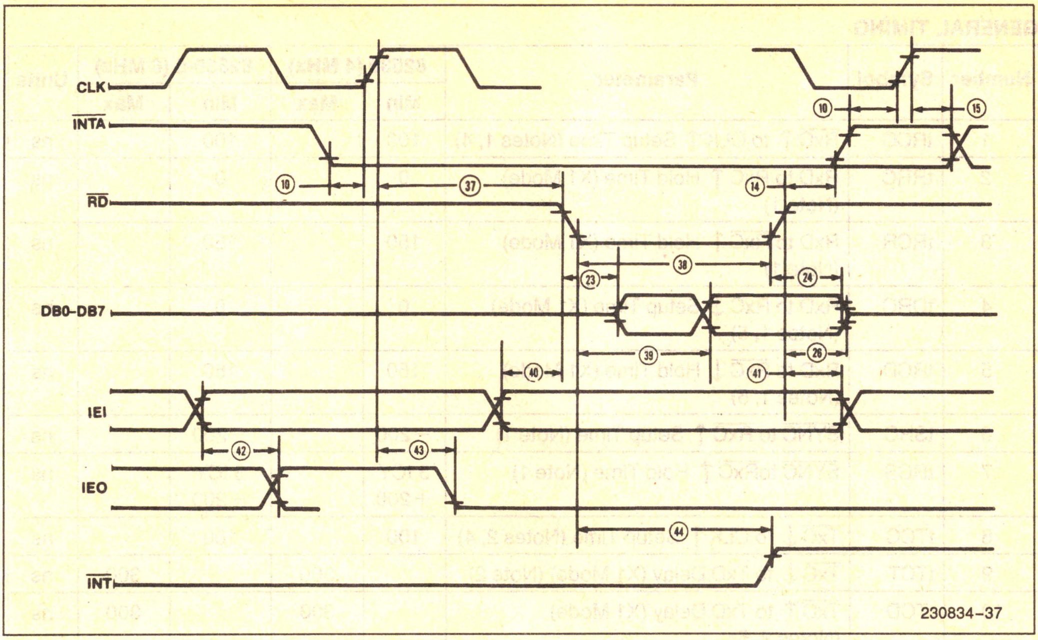


Figure 15. Interrupt Acknowledge Timing

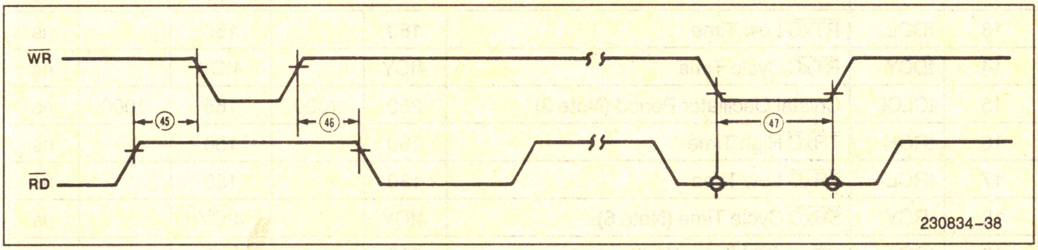


Figure 16. Reset Timing

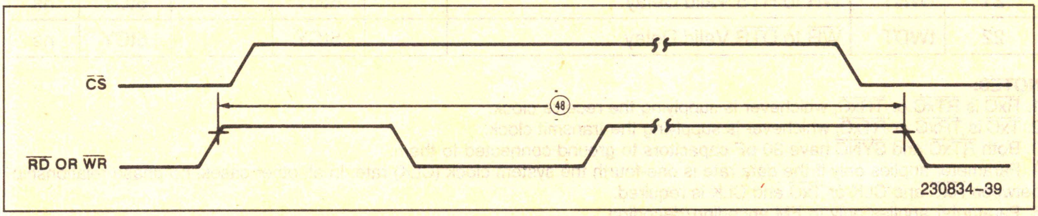


Figure 17. Cycle Timing

GENERAL TIMING

Number	Symbol	Parameter	82530 (4 MHz)		82530-6 (6 MHz)		Units
			Min	Max	Min	Max	
1	tRCC	$\overline{\text{RxC}}$ ↑ to CLK ↑ Setup Time (Notes 1, 4)	100		100		ns
2	tRRC	RxD to $\overline{\text{RxC}}$ ↑ Hold Time (X1 Mode) (Note 1)	0		0		ns
3	tRCR	RxD to $\overline{\text{RxC}}$ ↑ Hold Time (X1 Mode) (Note 1)	150		150		ns
4	tDRC	RxD to $\overline{\text{RxC}}$ ↓ Setup Time (X1 Mode) (Notes 1, 5)	0		0		ns
5	tRCD	RxD to $\overline{\text{RxC}}$ ↓ Hold Time (X1 Mode) (Notes 1, 5)	150		150		ns
6	tSRC	$\overline{\text{SYNC}}$ to $\overline{\text{RxC}}$ ↑ Setup Time (Note 1)	−200		−200		ns
7	tRCS	$\overline{\text{SYNC}}$ to$\overline{\text{RxC}}$ ↑ Hold Time (Note 1)	3 tCY +200		3 tCY +200		ns
8	tTCC	$\overline{\text{TxC}}$ ↓ to CLK ↑ Setup Time (Notes 2, 4)	100		100		ns
9	tTCT	$\overline{\text{TxC}}$ ↓ to TxD Delay (X1 Mode) (Note 2)		300		300	ns
10	tTCD	$\overline{\text{TxC}}$ ↑ to TxD Delay (X1 Mode) (Notes 2, 5)		300		300	ns
11	tTDT	TxD to $\overline{\text{TRxC}}$ Delay (Send Clock Echo)		200		200	ns
12	tDCH	$\overline{\text{RTxC}}$ High Time	180		150		ns
13	tDCL	$\overline{\text{RTxC}}$ Low Time	180		150		ns
14	tDCY	$\overline{\text{RTxC}}$ Cycle Time	4tCY		4tCY		ns
15	tCLCL	Crystal Oscillator Period (Note 3)	250	1000	165	1000	ns
16	tRCH	$\overline{\text{TRxC}}$ High Time	180		150		ns
17	tRCL	$\overline{\text{TRxC}}$ Low Time	180		150		ns
18	tRCY	$\overline{\text{TRxC}}$ Cycle Time (Note 6)	4tCY		4tCY		ns
19	tCC	$\overline{\text{CD}}$ or $\overline{\text{CTS}}$ Pulse Width	200		200		ns
20	tSS	$\overline{\text{SYNC}}$ Pulse Width	200		200		ns
21	tWRT	$\overline{\text{WR}}$ to $\overline{\text{RTS}}$ Valid Delay		6tCY		6tCY	ns
22	tWDT	$\overline{\text{WR}}$ to $\overline{\text{DTR}}$ Valid Delay		5tCY		5tCY	ns

NOTES:
1. $\overline{\text{RxC}}$ is $\overline{\text{RTxC}}$ or $\overline{\text{TRxC}}$, whichever is supplying the receive clock.
2. $\overline{\text{TxC}}$ is $\overline{\text{TRxC}}$ or $\overline{\text{RTxC}}$, whichever is supplying the transmit clock.
3. Both $\overline{\text{RTxC}}$ and $\overline{\text{SYNC}}$ have 30 pF capacitors to ground connected to them.
4. Parameter applies only if the data rate is one-fourth the system clock (CLK) rate. In all other cases, no phase relationship between $\overline{\text{RxC}}$ and CLK or $\overline{\text{TxC}}$ and CLK is required.
5. Parameter applies only to FM encoding/decoding.
6. Only applies to transmitter and receiver. For DPLL and Baud Rate Generator Timings, the requirements are identical to system clock, CLK, specifications.

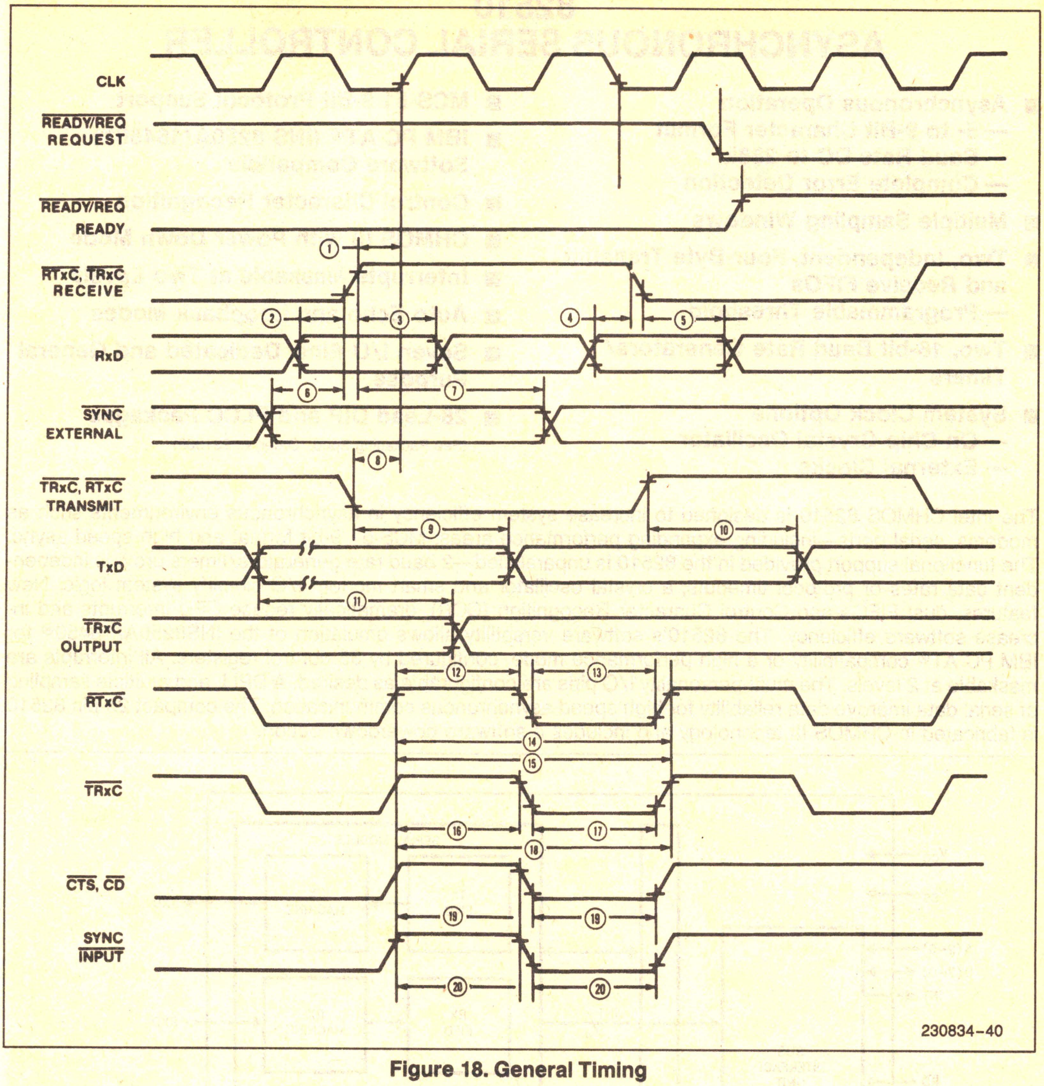


Figure 18. General Timing

82510 ASYNCHRONOUS SERIAL CONTROLLER

- Asynchronous Operation
  - 5- to 9-Bit Character Format
  - Baud Rate DC to 288k
  - Complete Error Detection
- Multiple Sampling Windows
- Two, Independent, Four-Byte Transmit and Receive FIFOs
  - Programmable Threshold
- Two, 16-bit Baud Rate Generators/ Timers
- System Clock Options
  - On-Chip Crystal Oscillator
  - External Clocks
- MCS-51 9-Bit Protocol Support
- IBM PC AT® (INS 8250A/16450®) Software Compatible
- Control Character Recognition
- CHMOS III with Power Down Mode
- Interrupts Maskable at Two Levels
- Auto Echo and Loopback Modes
- Seven I/O Pins, Dedicated and General Purpose
- 28-Lead DIP and PLCC Packages
  (See Packaging Spec., Order #: 231369)

The Intel CHMOS 82510 is designed to increase system efficiency in asynchronous environments such as modems, serial ports—including expanding performance areas: MCS-51 9-bit format and high speed async. The functional support provided in the 82510 is unparalleled—2 baud rate generators/timers provide independent data rates or protocol timeouts; a crystal oscillator and smart modem I/O simplify system logic. New features, dual FIFOs and Control Character Recognition (CCR), dramatically reduce CPU interrupts and increase software efficiency. The 82510's software versatility allows emulation of the INS8250A/16450® for IBM PC AT® compatibility or a high performance mode, configured by 35 control registers. All interrupts are maskable at 2 levels. The multi-personality I/O pins are configurable as desired. A DPLL and multiple sampling of serial data improve data reliability for high speed asynchronous communication. The compact 28-pin 82510 is fabricated in CHMOS III technology and includes a software powerdown option.

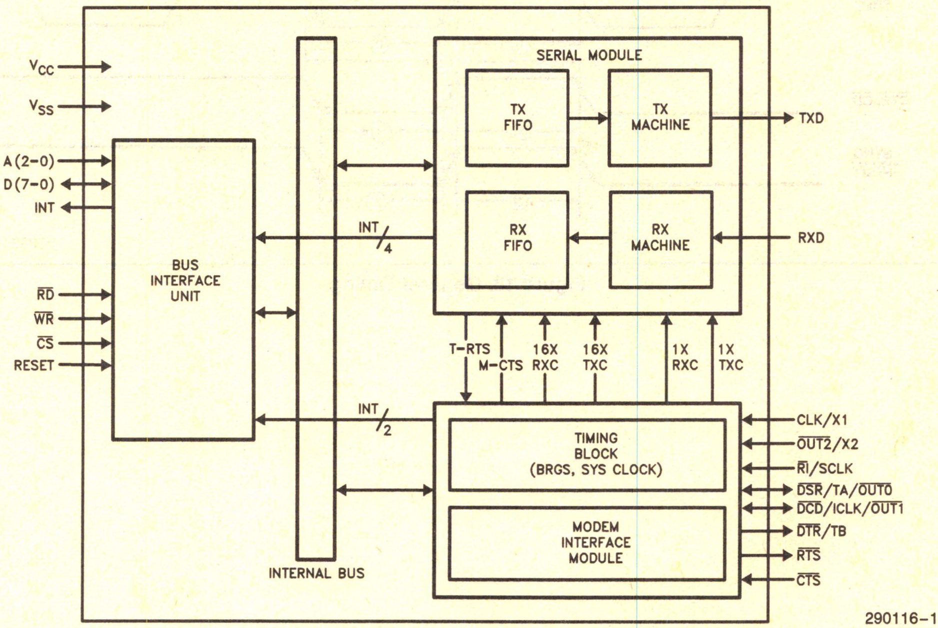


Figure 1. Block Diagram

December 1986
Order Number: 290116-002

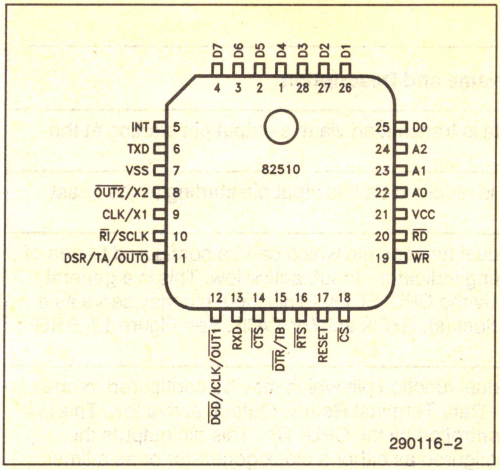


Figure 2. PLCC Pinout

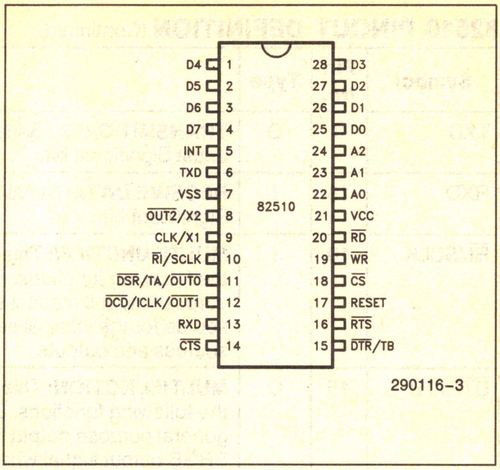


Figure 3. DIP Pinout

82510 PINOUT DEFINITION

Symbol	Pin No.	Type	Name and Description
RESET	17	I	**RESET:** A high on this input pin resets the 82510 to the Default Wake-up mode.
$\overline{CS}$	18	I	**CHIP SELECT:** A low on this input pin enables the 82510 and allows read or write operations.
A2–A0	24-22	I	**ADDRESS PINS:** These inputs interface with three bits of the System Address Bus to select one of the internal registers for read or write.
D7–D0	4* 25	I/O	**DATA BUS:** Bi-directional, three state, eight-bit Data Bus. These pins allow transfer of bytes between the CPU and the 82510.
$\overline{RD}$	20	I	**READ:** A low on this input pin allows the CPU to read Data or Status bytes from the 82510.
$\overline{WR}$	19	I	**WRITE:** A low on this input allows the CPU to write Data or Control bytes to the 82510.
INT	5	O	**INTERRUPT:** A high on this output pin signals an interrupt request to the CPU. The CPU may determine the particular source and cause of the interrupt by reading the 82510 Status registers.
CLK/X1	9	I	**MULTIFUNCTION:** This input pin serves as a source for the internal system clock. The clock may be asynchronous to the serial clocks and to the processor clock. This pin may be used in one of two modes: CLK — in this mode an externally generated TTL compatible clock should be used to drive this input pin; X1 — in this mode the clock is internally generated by an on-chip crystal oscillator. This mode requires a crystal to be connected between this pin (X1) and the X2 pin. (See System Clock Generation.)
$\overline{OUT2}$/X2	8	O	**MULTIFUNCTION:** This is a dual function pin which may be configured to one of the following functions: $\overline{OUT2}$ — a general purpose output pin controlled by the CPU, only available when CLK/X1 pin is driven by an externally generated clock; X2 - this pin serves as an output pin for the crystal oscillator. *Note*: The configuration of the pin is done only during hardware reset. For more details refer to the System Clock Generation.

*Pins 28–25 and Pins 4–1.

82510 PINOUT DEFINITION (Continued)

Symbol	Pin No.	Type	Name and Description
TXD	6	O	**TRANSMIT DATA:** Serial data is transmitted via this output pin starting at the Least Significant bit.
RXD	13	I	**RECEIVE DATA:** Serial data is received on this input pin starting at the Least Significant bit.
$\overline{\mathrm{RI}}$/SCLK	10	I	**MULTIFUNCTION:** This is a dual function pin which can be configured to one of the following functions. $\overline{\mathrm{RI}}$ - Ring Indicator - Input, active low. This is a general purpose input pin accessible by the CPU. SCLK - This input pin may serve as a source for the internal serial clock(s), RxClk and/or TxClk. See Figure 12, BRG sources and outputs.
$\overline{\mathrm{DTR}}$/TB	15	O	**MULTIFUNCTION:** This is a dual function pin which may be configured to one of the following functions. $\overline{\mathrm{DTR}}$ - Data Terminal Ready. Output, active low. This is a general purpose output pin controlled by the CPU. TB - This pin outputs the BRGB output signal when configured as either a clock generator or as a timer. When BRGB is configured as a timer this pin outputs a "timer expired pulse." When BRGB is configured as a clock generator it outputs the BRGB output clock.
$\overline{\mathrm{DSR}}$/TA/ $\overline{\mathrm{OUT0}}$	11	I/O	**MULTIFUNCTION:** This is a multifunction pin which may be configured to one of the following functions. $\overline{\mathrm{DSR}}$ - Data Set Ready. Input, active low. This is a general purpose input pin accessible by the CPU. TA - This pin is similar in function to pin TB except it outputs the signals from BRGA instead of BRGB. $\overline{\mathrm{OUT0}}$ - Output pin. This is a general purpose output pin controlled by the CPU.
$\overline{\mathrm{RTS}}$	16	O	**REQUEST TO SEND:** Output pin, active low. This is a general purpose output pin controlled by the CPU. In addition, in automatic transmission mode this pin, along with $\overline{\mathrm{CTS}}$, controls the transmission of data. (See Transmit modes for further detail.) During hardware reset this pin is an input. It is used to determine the System Clock Mode. (See System Clock Generation for further detail.)
$\overline{\mathrm{CTS}}$	14	I	**CLEAR TO SEND:** Input pin, active low. In automatic transmission mode it directly controls the Transmit Machine. (See transmission mode for further details.) This pin can be used as a General Purpose Input.
$\overline{\mathrm{DCD}}$/ICLK/ $\overline{\mathrm{OUT1}}$	12	I/O	**MULTIFUNCTION:** This is a multifunction pin which may be configured to one of the following functions. $\overline{\mathrm{DCD}}$ - Data Carrier Detected. Input pin, active low. This is a general purpose input pin accessible by the CPU. ICLK - This pin is the output of the internal system clock. $\overline{\mathrm{OUT1}}$ - General purpose output pin. Controlled by the CPU.

Table 1. Multifunction Pins

Pin #	I/O	Timing	Modem
8	*$\overline{\mathrm{OUT2}}$	X2	—
9	—	*CLK/X1	—
10	—	SCLK	*$\overline{\mathrm{RI}}$
11	$\overline{\mathrm{OUT0}}$	TA	*$\overline{\mathrm{DSR}}$
12	$\overline{\mathrm{OUT1}}$	ICLK	*$\overline{\mathrm{DCD}}$
14	—	—	*$\overline{\mathrm{CTS}}$
15	—	TB	*$\overline{\mathrm{DTR}}$
16	—	—	*$\overline{\mathrm{RTS}}$

*Default

GENERAL DESCRIPTION

The 82510 can be functionally divided into seven major blocks (See Fig 1): Bus Interface Unit, Timing Unit, Modem Module, Tx FIFO, Rx FIFO, Tx Machine, and Rx Machine. Six of these blocks (all except Bus Interface Unit) can generate block interrupts. Three of these blocks can generate second-level interrupts which reflect errors/status within the block (Receive Machine, Timing Unit, and the Modem Module).

The Bus interface unit allows the 82510 to interface with the rest of the system. It controls access to device registers as well as generation of interrupts to the external world. The FIFOs buffer the CPU from the Serial Machines and reduce the interrupt overhead normally required for serial operations. The threshold (level of occupancy in the FIFO which will generate an interrupt) is programmable for each FIFO. The timing unit controls generation of the system clock through either its on-chip crystal oscillator, or an externally generated clock. It also provides two Baud Rate Generators/Timers with various options and modes to support serial communication.

FUNCTIONAL DESCRIPTION

CPU Interface

The 82510 has a simple demultiplexed Bus Interface, which consists of a bidirectional three-state eight-bit, data bus and a three-bit address bus. An Interrupt pin along with the Read, Write and Chip Select are the remaining signals used to interface with the CPU. The three address lines along with the Bank Pointer register are used to select the registers. The 82510 is designed to interface to all Intel microprocessor and microcontroller families. Like most other I/O based peripherals it is programmed through its registers to support a variety of functions.

Its register set can be used in 8250A/16450 compatibility or High Performance modes. The 8250A/16450 mode is the default wake-up mode in which only the 8250A/16450 compatible registers are accessible. The remaining registers are default configured to support 8250A/16450 emulation.

Software Interface

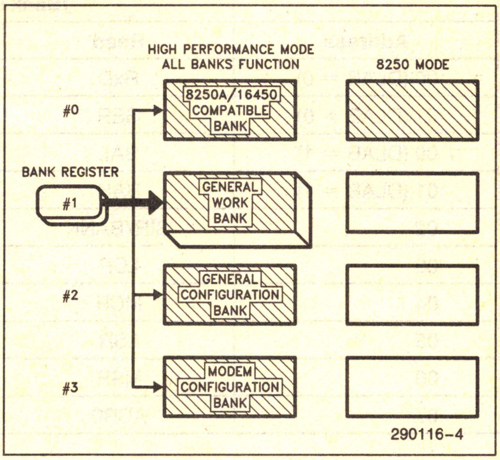


Figure 4. 82510 Register Architecture

The 82510 is configured and controlled through its 35 registers which are divided into four banks. Only one bank is accessible at any one time. The bank switching is done by changing the contents of the bank pointer (GIR/BANK-BANK0, BANK1). The banks are logically grouped into 8250A/16450 compatible (0), General Work Bank (1), General Configuration (2), and Modem Configuration (3). The 8250A/16450 compatible bank (Bank 0) is the default bank upon power up.

The 82510 registers can be categorized under the following:

Table 2. 82510 Register/Block Functions

	Status	Enable	Configuration	Command	Data
FIFO	FLR	—	FMD	—	—
MODEM	MSR	MIE	PMD	MCR	—
RX	RST, RXF	RIE	RMD	RCM	RXD, RXF
TX	LSR	LSR	TMD	TCM	TXD, TXF
TIMER	TMST	TMIE	CLCF, BACF, BBCF	TMCR	BBL, BBH BAL, BAH
DEVICE	GSR, GIR	GER	IMD	ICM	—
8250	LSR, MSR, GIR	GER	LCR, MCR	MCR	TXD, RXD BAL, BAH

8250 Compatibility

Upon power up or reset, the 82510 comes up in the default wake up mode. The 8250A/16450 compatible bank, bank zero, is the accessible bank and all the other registers are configured via their default values to support this mode.

Table 3. 8250A/16450 Compatible Registers

	82510 Registers (Bank 0)		8250A Registers	
Address	**Read**	**Write**	**Read**	**Write**
00 (DLAB = 0)	RxD	TxD	RBR	THR
01 (DLAB = 0)	GER	GER	IER	IER
00 (DLAB = 1)	BAL	BAL	DLL	DLL
01 (DLAB = 1)	BAH	BAH	DLM	DLM
02	GIR/BANK	BANK	IIR	—
03	LCR	LCR	LCR	LCR
04	MCR	MCR	MCR	MCR
05	LSR	LSR	LSR	LSR
06	MSR	MSR	MSR	MSR
07	ACR0	ACR0	SCR	SCR

Table 4. Default Wake-Up Mode

RxD	—	ACR1	00H	RxF	—
TxD	—	RIE	1EH	TxF	—
BAL	02H	RMD	00H	TMST	30H
BAH	00H	CLCF	00H	TMCR	—
GER	00H	BACF	04H	FLR	00H
GIR/BANK	01H	BBCF	84H	RCM	—
LCR	00H	PMD	FCH	TCM	—
MCR	00H	MIE	0FH	GSR	12H
LSR	60H	TMIE	00H	ICM	—
MSR	00H	BBL	05H	FMD	00H
ACR0	00H	BBH	00H	TMD	00H
RST	00H			IMD	0CH

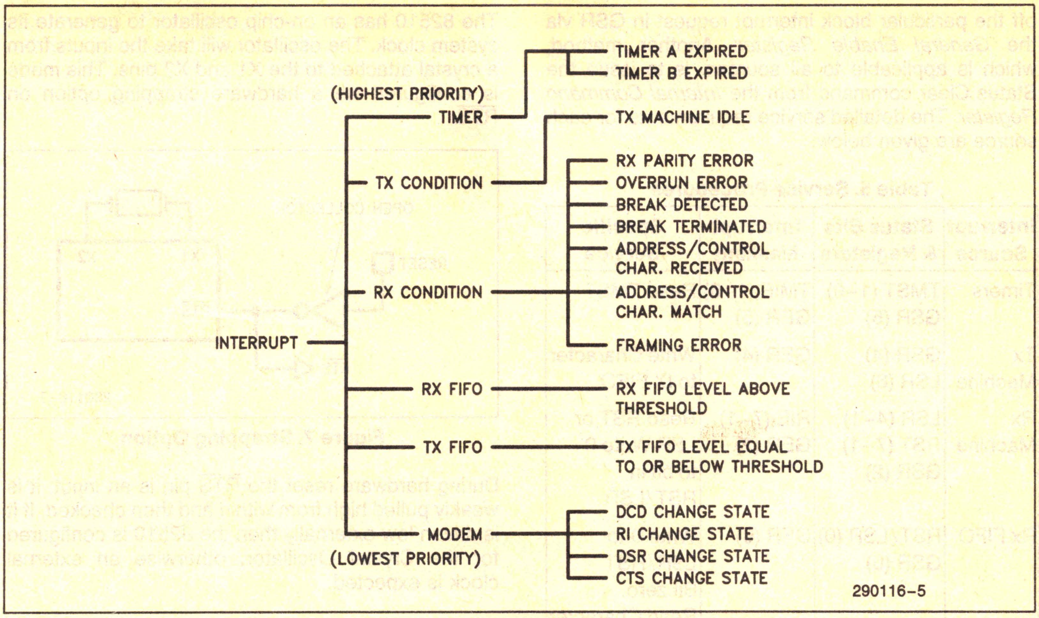


Figure 5. Interrupt Structure

Interrupts

There are two levels of interrupt/status reporting within the 82510. The first level is the block level interrupts such as RX FIFO, Tx FIFO, Rx Machine, Tx Machine, Timing unit, and Modem Module. The status of these blocks is reported in the General Status and General Interrupt Registers. The second level is the various sources within each block; only three of the blocks generate second level interrupts (Rx Machine, Timing Unit, and Modem Module). Interrupt requests are maskable at both the block level and at the individual source level within the module. If more than one unmasked block requests interrupt service an on-chip interrupt controller will resolve contention on a priority basis (each block has a fixed priority). An interrupt request from a particular block is activated if one of the unmasked status bits within the status register for the block is set. A CPU service operation, e.g., reading the appropriate status register, will reset the status bits.

ACKNOWLEDGE MODES

The interrupt logic will assert the INT pin when an interrupt is coded into the General Interrupt register. The INT pin is forced low upon acknowledgment. The 82510 has two modes of interrupt acknowledgment:

1. Manual Acknowledge

The CPU must issue an explicit Interrupt Acknowledge command via the Interrupt Acknowledge bit of the Internal Command register. As a result the INT pin is forced low for two clocks and then updated.

2. Automatic Acknowledge

As opposed to the Manual Acknowledge mode, when the CPU must issue an explicit interrupt acknowledge command, an interrupt service operation is considered as an automatic acknowledgment. This forces the INT pin low for two clock cycles. After two cycles the INT pin is updated, i.e., if there is still an active non-masked interrupt request the INT pin is set HIGH.

INTERRUPT SERVICE

A service operation is an operation performed by the CPU, which causes the source of the 82510 interrupt to be reset (it will reset the particular status bit causing the interrupt). An interrupt request within the 82510 will not reset until the interrupt source has been serviced. Each source can be serviced in two or three different ways; one general way is to disable the particular status bit causing the interrupt, via the corresponding block enable register. Setting the appropriate bit of the enable register to zero will mask off the corresponding bit in the status register, thus causing an edge on the input line to the interrupt logic. The same effect can be achieved by masking

off the particular block interrupt request in GSR via the *General Enable Register.* Another method, which is applicable to all sources, is to issue the Status Clear command from the *Internal Command Register.* The detailed service requirements for each source are given below:

Table 5. Service Procedures

Interrupt Source	Status Bits & Registers	Interrupt Masking	Specific Service
Timers	TMST (1–0) GSR (5)	TMIE (1–0) GER (5)	Read TMST
Tx Machine	GSR (4) LSR (6)	GER (4)	Write Character to tX FIFO
Rx Machine	LSR (4–1) RST (7–1) GSR (2)	RIE (7–1) GER (2)	Read RST or LSR Write 0 to bit in RST/LSR
Rx FIFO	RST/LSR (0) GSR (0)	GER (0)	Write 0 to LSR/RST Bit zero. Read Character
Tx FIFO	LSR (5) GSR (1)	GER (1)	Write to FIFO Read GIR(1)
Modem	MSR (3-0) GSR (3)	MIE (3-0) GER (3)	Read MSR write 0 into the appropriate bits of MSR (3–0).

NOTE:
1. Only if pending interrupt is Tx FIFO.

System Clock Generation

The 82510 has two modes of System Clock Operation. It can accept an externally generated clock, or it can use a crystal to internally generate its system clock.

CRYSTAL OSCILLATOR

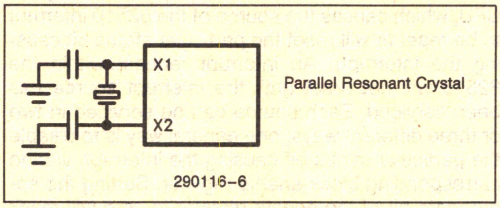


Figure 6. Crystal Oscillator

The 82510 has an on-chip oscillator to generate its system clock. The oscillator will take the inputs from a crystal attached to the X1 and X2 pins. This mode is configured via a hardware strapping option on $\overline{\text{RTS}}$.

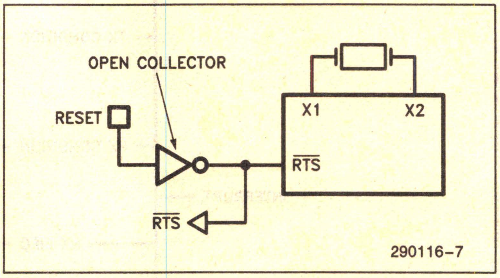


Figure 7. Strapping Option

During hardware reset the $\overline{\text{RTS}}$ pin is an input; it is weakly pulled high from within and then checked. If it is driven low externally then the 82510 is configured for the Crystal Oscillator; otherwise an external clock is expected.

EXTERNALLY GENERATED SYSTEM CLOCK

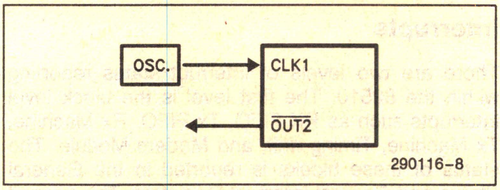


Figure 8. External Clock

This is the default configuration. Under normal conditions the system clock is divided by two; however, the user may disable divide by two via a hardware strapping option on the $\overline{\text{DTR}}$ pin. The Hardware strapping option is similar to the one used on the $\overline{\text{RTS}}$ pin. It is forbidden to strap both $\overline{\text{DTR}}$ and $\overline{\text{RTS}}$.

Transmit

The two major blocks involved in transmission are the Transmit FIFO and the Transmit Machine. The Tx FIFO acts as a buffer between the CPU and the Tx Machine. Whenever a data character is written to the Transmit Data register, it, along with the Transmit Flags (if applicable), is loaded into the Tx FIFO.

TX FIFO

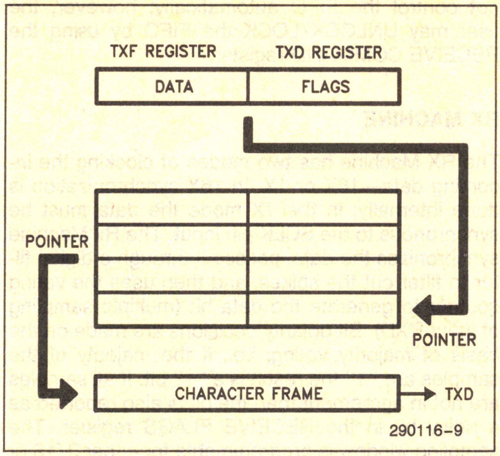


Figure 9. Tx FIFO

The Tx FIFO can hold up to four, eleven-bit characters (nine-bits data, parity, and address flag). It has separate read and write mechanisms. The read and write pointers are incremented after every operation to allow data transfer to occur in a First In First Out fashion. The Tx FIFO will generate a maskable interrupt when the level in the FIFO is below, or equal to, the Threshold. The threshold is user programmable.

For example, if the threshold equals two, and the number of characters in the Tx FIFO decreases from three to two, the FIFO will generate an interrupt. The threshold should be selected with regard to the system's interrupt service latency.

NOTE:
There is a one character transmission delay between FIFO empty and Transmitter Idle, so a threshold of zero may be selected without getting an underrun condition. Also if more than four characters are written to the FIFO an overrun will occur and the extra character will not be written to the Tx FIFO. This error will not be reported to the CPU.

TX MACHINE

The Tx Machine reads characters from the Tx FIFO, serializes the bits, and transmits them over the TXD pin according to the timing signals provided for transmission. It will also generate parity, transmit break (upon CPU request), and manage the modem handshaking signals ($\overline{CTS}$ and $\overline{RTS}$) if configured so. The Tx machine can be enabled or disabled through the Transmit Command register or $\overline{CTS}$. If the transmitter is disabled in the middle of a character transmission the transmission will continue until the end of the character; only then will it enter the disable state.

TRANSMIT CLOCKS

There are two modes of transmission clocking, 1X and 16X. In the 1X mode the transmitted data is synchronous to the transmit clock as supplied by the SCLK pin. In this mode stop-bit length is restricted to one or two bits only. In the 16X mode the data is not required to be synchronous to the clock. (Note: The Tx clock can be generated by the BRGs or from the SCLK pin.)

MODEM HANDSHAKING

The transmitter has three modes of handshaking.

Manual Mode—In this mode the $\overline{CTS}$ and $\overline{RTS}$ pins are not used by the Tx Machine (transmission is started regardless of the $\overline{CTS}$ state, and $\overline{RTS}$ is not forced low). The CPU may manage the handshake itself, by accessing the $\overline{CTS}$ and $\overline{RTS}$ signals through the MODEM CONTROL and MODEM STATUS registers.

Semi-Automatic Mode—In this mode the $\overline{RTS}$ pin is activated whenever the transmitter is enabled. The $\overline{CTS}$ pin's state controls transmission. Transmission is enabled only if $\overline{CTS}$ is active. If $\overline{CTS}$ becomes inactive during transmission, the Tx Machine will complete transmission of the current character and then go to the inactive state until $\overline{CTS}$ becomes active again.

Automatic Mode—This mode is similar to the semi-automatic mode, except that $\overline{RTS}$ will be activated as long as the transmitter is enabled and there are more characters to transmit. The CPU need only fill the FIFO, the handshake is done by the Tx Machine. When both the shift register and the FIFO are empty $\overline{RTS}$ automatically goes inactive. (Note: The $\overline{RTS}$ pin can be forced to the active state by the CPU, regardless of the handshaking mode, via the MODEM CONTROL register.)

Receive

The 82510 reception mechanism involves two major blocks; the Rx Machine and the Rx FIFO. The Rx Machine will assemble the incoming character and its associated flags and then LOAD them on to the Rx FIFO. The top of the FIFO may be read by reading the Receive Data register and the Receive Flags Register. The receive operation can be done in two modes. In the *normal* mode the characters are received in the standard Asynchronous format and only control characters are recognized. In the *ulan* mode, the nine bit protocol of the MCS-51 family is supported and the ulan Address characters, rather than Control Characters are recognized.

RX FIFO

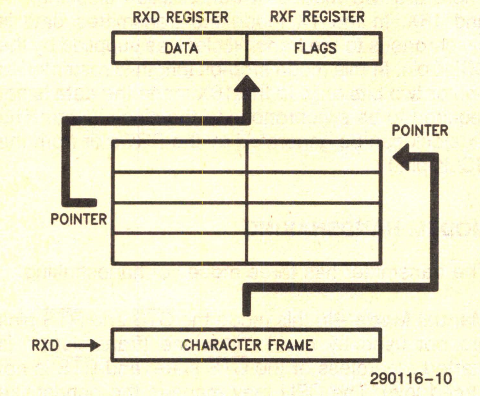


Figure 10. Rx FIFO

The Rx FIFO is very similar in structure and basic operation to the Tx FIFO. It will generate a maskable interrupt when the FIFO level is above, the threshold. The Rx FIFO can also be configured to operate as a one-byte buffer. This mode is used for 8250 compatible software drivers. An overrun will occur when the FIFO is full and the Rx Machine has a new character for the FIFO. In this situation the oldest character is discarded and the new character is loaded from the Rx Machine. An Overrun error bit will also be set in the RECEIVE STATUS and LINE STATUS registers.

The user has the option to disable the loading of incoming characters on to the Rx FIFO by using the UNLOCK/LOCK FIFO commands. (See RECEIVE COMMAND register.) When the Rx FIFO is locked, it will ignore load requests from the Rx Machine, and thus the received characters will not be loaded into the FIFO and may be lost (if another character is received). These two commands are useful when the CPU is not willing to receive characters, or is waiting for specific Control/Address characters. In uLAN mode there are three options of address recognition, each of these options varies in the amount of CPU offload, and degree of FIFO control through OPEN/LOCK FIFO commands.

Automatic Mode—In this mode the Rx Machine will open the FIFO whenever an Address Match occurs; it will LOCK the FIFO if an address mismatch occurs.

Semi-Automatic Mode—In this mode the Rx Machine will open the FIFO whenever an address character is received. It will not lock the FIFO if the Address does not match. The user is responsible for locking the Rx FIFO.

Manual Mode—In this mode the Rx Machine does not control the FIFO automatically; however, the user may UNLOCK/LOCK the FIFO by using the RECEIVE COMMAND register.

RX MACHINE

The RX Machine has two modes of clocking the incoming data—16X or 1X. In 16X synchronization is done internally; in the 1X mode the data must be synchronous to the SCLK pin input. The Rx Machine synchronizes the data, passes it through a digital filter to filter out the spikes, and then uses the voting counter to generate the data bit (multiple sampling of input RXD). Bit polarity decisions are made on the basis of majority voting; i.e., if the majority of the samples are "1" the result is a "1" bit. If all samples are not in agreement then the bit is also reported as a noisy bit in the RECEIVE FLAGS register. The sampling window is programmable for either 3/16 or 7/16 samples. The 3/16 mode is useful for high frequency transmissions, or when serious RC delays are expected on the channel. The 7/16 is best suited for noisy media. The Rx machine also has a DPLL to overcome frequency shift problems; however, using it in a very noisy environment may increase the error, so the user can disable the DPLL via the Receive Mode register. The Rx Machine will generate the parity and the address marker as well as any framing error indications.

Start Bit Detection—The falling edge of the Start bit resets the DPLL counter and the Rx Machine starts sampling the input line (the number of samples is determined by the configuration of the sampling window mode). The Start bit verification can be done through either a majority voting system or an absolute voting system. The absolute voting requires that all the samples be in agreement. If one of the samples does not agree then a false Start bit is determined and the Rx Machine returns to the Start Bit search Mode. Once a Start bit is detected the Rx Machine will use the majority voting sampling window to receive the data bits.

Break Detection—If the input is low for the entire character frame including the stop Bit, then the Rx Machine will set Break Detected as well as Framing Error in the RECEIVE STATUS and LINE STATUS registers. It will push a NULL character onto the Rx FIFO with a framing-error and Break flag (As part of the Receive Flags). The Rx Machine then enters the Idle state. When it sees a mark it will set Break Terminated in RECEIVE STATUS and LINE STATUS registers and resume normal operation.

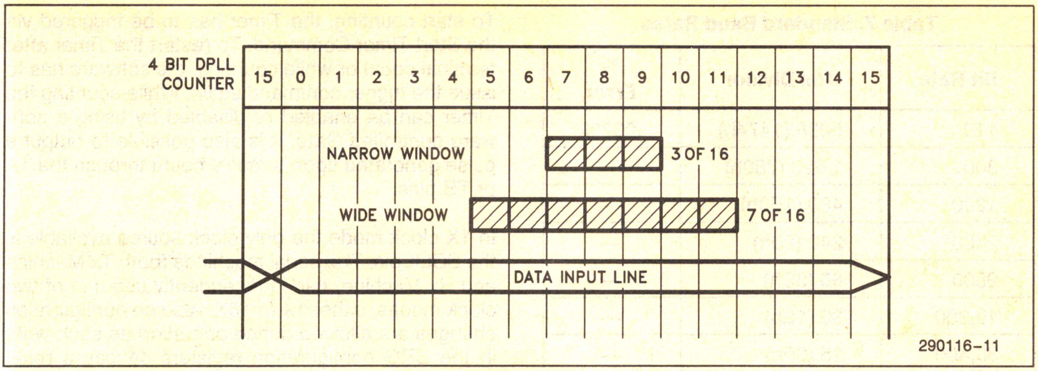


Figure 11. Sampling Windows

Control Characters—The Rx machine can generate a maskable interrupt upon reception of standard ASCII or EBCDIC control characters, or an Address marker is received in the uLAN mode. The Rx machine can also generate a maskable interrupt upon a match with programmed characters in the Address/Control Character 0 or Address/Control Character 1 registers.

Table 6. Control Character Recognition

CONTROL CHARACTER RECOGNITION		
A}	STANDARD SET	
	■ ASCII:	000X XXXX + 0111 1111 (ASCII DEL) (00 - 1FH + 7 FH)
	OR	
	■ EBCDIC:	00XX XXXX (00 - 3FH)
B}	User Programmed	
	■ ACR0, ACR1	XXXX XXXX REGISTERS

Baud-Rate Generators/Timers

The 82510 has two-on-chip, 16-bit baud-rate generators. Each BRG can also be configured as a Timer, and is completely independent of the other. This can be used when the Transmit and Receive baud rates are different. The mode, the output, and the source of each BRG is configurable, and can also be optionally output to external devices via the TA, TB pins (see Fig. 12. BRG Sources and Outputs).

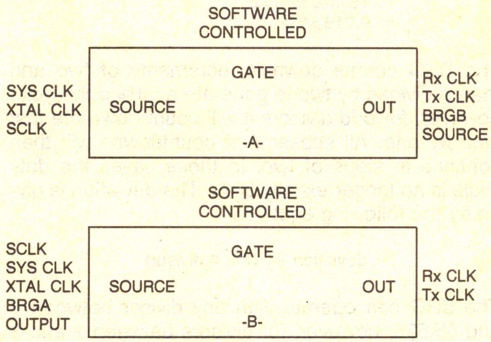


Figure 12. BRG Sources and Outputs

BAUD RATE GENERATION

The Baud Rate is generated by dividing the source clock with the divisor count (from the Divisor count registers). The count is loaded from the divisor count registers into a count down register. A 50% duty cycle is generated by counting down in steps of two. When the count is down to 2 the entire count is reloaded and the output clock is toggled. Optionally the two BRGs may be cascaded to provide a larger divisor.

$$f_O = f_{in}/\mathrm{Divisor}$$

where $f$in is the input clock frequency and Divisor is the count loaded into the appropriate count registers.

Table 7. Standard Baud Rates

Bit Rate	16x Divisor	% Error
110	5236 (1474h)	.007%
300	1,920 (780h)	—
1200	480 (1E0h)	—
2400	240 (F0h)	—
9600	60 (3Ch)	—
19,200	30 (1Eh)	—
38,400	15 (0Fh)	—
56,000	10 (0Ah)	2.8%
288,000	2 (02h)	—

Source CLK = Internal Sys. Clk
= 18.432 MHz/2
= 9.216 MHz

The BRG counts down in increments of two and then is divided by two to generate a 50% duty cycle; however, for odd divisors it will count down the first time by one. All subsequent countdowns will then continue in steps of two. In those cases the duty cycle is no longer exactly 50%. The deviation is given by the following equation:

$$\text{deviation} = 1/(2 \bullet \text{divisor})$$

The BRG can operate with any divisor between 1 and 65,535; however, for divisors between 1 and 3 the duty cycle is as follows:

Table 8. Duty Cycles

Divisor	Duty Cycle
3	33%
2	50%
1	Same as Source
0	FORBIDDEN

Timer Mode

Each of the 82510 BRGs can be used as Timers. The Timer is used to generate time delays by counting the internal system clock. When enabled the Timer uses the count from the Divisor/Count registers to count down to 1. Upon terminal count a maskable Timer Expired interrupt is generated. The delay between the trigger and the terminal count is given by the following equation:

$$\text{Delay} = \text{Count} \bullet \text{(System Clock Period)}$$

To start counting, the Timer has to be triggered via the Start Timer Command. To restart the Timer after terminal count or while counting, the software has to issue the trigger command again. While counting the Timer can be enabled or disabled by using a software controlled Gate. It is also possible to output a pulse generated upon terminal count through the TA or TB pins.

In 1X clock mode the only clock source available is the SCLK pin. The serial machines (both Tx Machine and Rx Machine) can independently use one of two clock modes, either 1X or 16X. Also no configuration changes are allowed during operation as each write in the BRG configuration registers causes a reset signal to be sent to the BRG logic. The mode or source clocks may be changed only after a Hardware or Software reset. The Divisor (or count, depending upon the mode) may be updated during operation unless the particular BRG machine is being used as a clock source for one of the serial machines, and the particular serial machine is in operation at the time. Loading the count registers with "0" is forbidden in all cases, and loading it with a "1" is forbidden in the Timer Mode only.

SERIAL DIAGNOSTICS

The 82510 supports two modes of Loopback operation, Local Loopback and Remote Loopback as well as an Echo mode for diagnostics and improved throughput.

LOCAL LOOPBACK

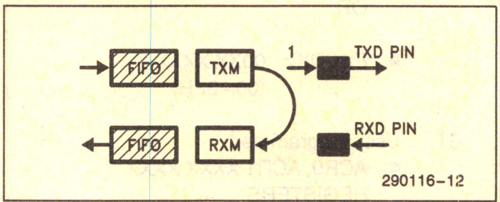


Figure 13. Local Loopback

The Tx Machine output and Rx Machine input are shorted internally, TXD pin output is held at Mark. This feature allows simulation of Transmission/Reception of characters and checks the Tx FIFO, Tx Machine, Rx Machine, and Rx FIFO along with the software without any external side effects. The modem outputs $\overline{OUT1}$, $\overline{OUT2}$, $\overline{DTR}$ and $\overline{RTS}$ are internally shorted to $\overline{RI}$, $\overline{DCD}$, $\overline{DSR}$ and $\overline{CTS}$ respectively. $\overline{OUT0}$ is held at a mark state.

REMOTE LOOPBACK

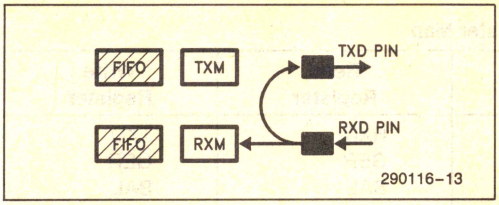


Figure 14. Remote Loopback

The TXD pin and RXD pin are shorted internally (the data is not sent on to the RX Machine). This feature allows the user to check the communications channel as well as the Tx and Rx pin circuits not checked in the Local Loopback mode.

AUTO ECHO

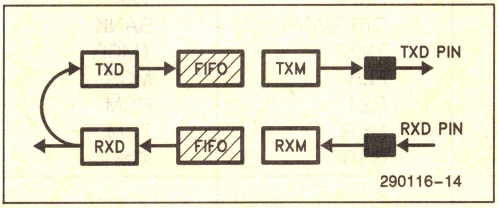


Figure 15. Auto Echo

In Echo Mode the received characters are automatically transmitted back. When the characters are read from the Rx FIFO they are automatically pushed back onto the Tx FIFO (the flags are also included). The Rx Machine baud rate must be equal to, or less than, the Tx Machine baud rate or some of the characters may be lost. The user has an option of preventing echo of special characters; Control Characters and characters with Errors.

Power Down Mode

The 82510 has a "power down" mode to reduce power consumption when the device is not in use. The 82510 powers down when the power down command is issued via the Internal Command Register (ICM). There are two modes of power down, Sleep and Idle.

In Sleep mode, even the system clock of the 82510 is shut down. The system clock source of the 82510 can either be the Crystal Oscillator or an external clock source. If the Crystal Oscillator is being used and the power down command is issued, then the 82510 will automatically enter the Sleep mode. If an external clock is being used, then the user must disable the external clock in addition to issuing the Power Down command, to enter the Sleep mode. The benefit of this mode is the increased savings in power consumption (typical power consumption in the Sleep mode is in the ranges of 100s of microAmps). However, upon wake up, the user must reprogram the device. To exit this mode the user can either issue a Hardware reset, or read the FIFO Level Register (FLR) and then issue a software reset. In either case the contents of the 82510 registers are not preserved and the device must be reprogrammed prior to operation. If the Crystal Oscillator is being used then the user must allow enough time for the oscillator to wake up before issuing the software reset.

The 82510 is in the idle mode when the Power Down command is issued and the system clock is still running (i. e. the system clock is generated externally and not disabled by the user). In this mode the contents of all registers and memory cells are preserved, however, the power consumption in this mode is greater than in the Sleep mode. Reading FLR will take the 82510 out of this mode.

NOTE:
The data read from FLR when exiting Power Down is invalid and should be ignored.

DETAILED REGISTER DESCRIPTION

Table 9. Register Map

Bank	Address	Read Register	Write Register
0 (NAS) 8250A/16450	0 (DLAB = 0)	RXD	TXD
	1 (DLAB = 0)	GER	GER
	0 (DLAB = 1)	BAL	BAL
	1 (DLAB = 1)	BAH	BAH
	2	GIR/BANK	BANK
	3	LCR	LCR
	4	MCR	MCR
	5	LSR	LSR
	6	MSR	MSR
	7	ACR0	ACR0
1 (WORK)	0	RXD	TXD
	1	RXF	TXF
	2	GIR/BANK	BANK
	3	TMST	TMCR
	4	FLR	MCR
	5	RST	RCM
	6	MSR	TCM
	7	GSR	ICM
2 (GENERAL CONF)	0	—	—
	1	FMD	FMD
	2	GIR/BANK	BANK
	3	TMD	TMD
	4	IMD	IMD
	5	ACR1	ACR1
	6	RIE	RIE
	7	RMD	RMD
3 (MODEM CONF)	0 (DLAB = 0)	CLCF	CLCF
	1 (DLAB = 0)	BACF	BACF
	0 (DLAB = 1)	BBL	BBL
	1 (DLAB = 1)	BBH	BBH
	2	GIR/BANK	BANK
	3	BBCF	BBCF
	4	PMD	PMD
	5	MIE	MIE
	6	TMIE	TMIE
	7	—	—

(1) ACRO is used in INS8250 as a Scratch-Pad Register
(2) DLAB = LCR Bit #7

The 82510 has thirty-five registers which are divided into four banks of register banks. Only one bank is accessible at any one time. The bank is selected through the BANK1-0 bits in the GIR/BANK register. The individual registers within a bank are selected by the three address lines (A2-0). The 82510 registers can be grouped into the following categories.

BANK ZERO 8250A/16450—COMPATIBLE BANK										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
TxD (33)	Tx Data bit 7	Tx Data bit 6	Tx Data bit 5	Tx Data bit 4	Tx Data bit 3	Tx Data bit 2	Tx Data bit 1	Tx Data bit 0	0	—
RxD (35)	Rx Data bit 7	Rx Data bit 6	Rx Data bit 5	Rx Data bit 4	Rx Data bit 3	Rx Data bit 2	Rx Data bit 1	Rx Data bit 0	0	—
BAL (11)	BRGA LSB Divide Count (DLAB = 1)								0	02H
BAH (12)	BRGA MSB Divide Count (DLAB = 1)								1	00H
GER (16)	0	0	Timer Interrupt Enable	Tx Machine Interrupt Enable	Modem Interrupt Enable	Rx Machine Interrupt Enable	Tx FIFO Interrupt Enable	Rx FIFO Interrupt Enable	1	00H
GIR/BANK (21)	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
LCR (2)	DLAB Divisor Latch Access bit	Set Break	Parity Mode bit 2	Parity Mode bit 1	Parity Mode bit 0	Stop bit Length bit 0	Character Length bit 1	Character Length bit 0	3	00H
MCR (32)	0	0	OUT 0 Complement	Loopback Control bit	OUT 2 Complement	OUT 1 Complement	RTS Complement	DTR Complement	4	00H
LSR (22)	0	TxM Idle	Tx FIFO Interrupt	Break Detected	Framing Error	Parity Error	Overrun Error	Rx FIFO Int Req	5	60H
MSR (27)	DCD Input Inverted	RI Input Inverted	DSR Input Inverted	CTS Input Inverted	State Change in DCD	State (H → L) Change in RI	State Change in DSR	State Change in CTS	6	00H
ACR0 (5)	Address or Control Character Zero								7	00H

BANK ONE—GENERAL WORK BANK										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
TxD (33)	Tx Data bit 7	Tx Data bit 6	Tx Data bit 5	Tx Data bit 4	Tx Data bit 3	Tx Data bit 2	Tx Data bit 1	Tx Data bit 0	0	—
RxD (35)	Rx Data bit 7	Rx Data bit 6	Rx Data bit 5	Rx Data bit 4	Rx Data bit 3	Rx Data bit 2	Rx Data bit 1	Rx Data bit 0	0	—
RxF (24)	—	Rx Char OK	Rx Char Noisy	Rx Char Parity Error	Address or Control Character	Break Flag	Rx Char Framing Error	Ninth Data bit of Rx Char	1	—
TxF (34)	Address Marker bit	Software Parity bit	Ninth bit of Data Char	0	0	0	0	0	1	—
GIR/BANK (21)	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
TMST (26)	—	—	Gate B State	Gate A State	—	—	Timer B Expired	Timer A Expired	3	30H
TMCR (31)	0	0	Trigger Gate B	Trigger Gate A	0	0	Start Timer B	Start Timer A	3	—
MCR (32)	0	0	OUT 0 Complement	Loopback Control bit	OUT 2 Complement	OUT 1 Complement	RTS Complement	DTR Complement	4	00H

NOTE:
The register number is provided as a reference number for the register description.

BANK ONE—GENERAL WORK BANK (Continued)										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
FLR (25)	—	Rx FIFO Level			—	Tx FIFO Level			4	00H
RST (23)	Address/ Control Character Received	Address/ Control Character Match	Break Terminated	Break Detected	Framing Error	Parity Error	Overrun Error	Rx FIFO Interrupt Requested	5	00H
RCM (30)	Rx Enable	Rx Disable	Flush RxM	Flush Rx FIFO	Lock Rx FIFO	Open Rx FIFO	0	0	5	—
MSR (27)	DCD Complement	RI Input Inverted	DSR Input Inverted	CTS Input Inverted	State Change in DCD	State Change in RI	State Change in DSR	State Change in CTS	6	00H
TCM (29)	0	0	0	0	Flush Tx Machine	Flush Tx FIFO	Tx Enable	Tx Disable	6	—
GSR (20)	—	—	Timer Interrupt	TxM Interrupt	Modem Interrupt	RxM Interrupt	Tx FIFO Interrupt	Rx FIFO Interrupt	7	12H
ICM (28)	0	0	0	Software Reset	Manual Int Acknowledge Command	Status Clear	Power Down Mode	0	7	—

BANK TWO—GENERAL CONFIGURATION										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
FMD (4)	0	0	Rx FIFO Threshold		0	0	Tx FIFO Threshold		1	00H
GIR/BANK (21)	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
TMD (3)	Error Echo Disable	Control Character Echo Disable	9-bit Character Length	Transmit Mode		Software Parity Mode	Stop Bit Length		3	00H
IMD (1)	0	0	0	0	Interrupt Acknowledge Mode	Rx FIFO Depth	ulan Mode Select	Loopback or Echo Mode of Operation	4	0CH
ACR1 (6)	Address or Control Character 1								5	00H
RIE (17)	Address/ Control Character Recognition Interrupt Enable	Address/ Control Character Match Interrupt Enable	Break Terminate Interrupt Enable	Break Detect Interrupt Enable	Framing Error Interrupt Enable	Parity Error Interrupt Enable	Overrun Error Interrupt Enable	0	6	1EH
RMD (7)	Address/Control Character Mode		Disable DPLL	Sampling Window Mode	Start bit Sampling Mode	0	0	0	7	00H

BANK THREE—MODEM CONFIGURATION										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
CLCF (8)	Rx Clock Mode	Rx Clock Source	Tx Clock Mode	Tx Clock Source	0	0	0	0	0	00H
BACF (9)	0	BRGA Clock Source	0	0	0	BRGA Mode	0	0	1	04H
BBL (13)	BRGB LSB Divide Count (DLAB = 1)								0	05H
BBH (14)	BRGB MSB Divide Count (DLAB = 1)								1	00H

BANK THREE—MODEM CONFIGURATION (Continued)										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
GIR/BANK (21)	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
BBCF (10)	BRGB Clock Source		0	0	0	BRGB Mode	0	0	3	84H
PMD (15)	DCD/ICLK/OUT 1 Direction	DCD/ICLK/OUT 1 Function	DSR/TA/OUT 0 Direction	DSR/TA/OUT 0 Function	RI/SCLK Function	DTR/TB Function	0	0	4	FCH
MIE (19)	0	0	0	0	DCD State Change Int Enable	RI State Change Int Enable	DSR State Change Int Enable	CTS State Change Int Enable	5	0FH
TMIE (18)	0	0	0	0	0	0	Timer B Interrupt Enable	Timer A Interrupt Enable	6	00H

CONFIGURATION

These read/write registers are used to configure the device. They may be read at anytime; however, they may be written to only when the device is idle. Typically they are written to only once after system power up. They are set to default values upon Hardware or Software Reset (Default Wake-Up Mode). The default values are chosen so as to allow the 82510 to be fully software compatible with the IBM PC Async Adapter (INS 8250A/16450) when in the default wakeup mode. The 82510 can operate in the High Performance mode by programming the configuration registers as necessary.

The configuration options available to the user are listed below.

Table 11. Configuration Options

Interrupt Acknowledge Mode
- Automatic
- Manual

Receive
- Sampling Window Size
- Start Bit Detection Mode
- DPLL Disable/Enable

μLAN (8051)
Address Recognition
- Manual, Semi-Automatic, Automatic

Diagnostics
- Loopback
  - Remote
  - Local
- Echo
  - Yes/No
  - Disable Error Echo
  - Disable Control/Address Char. Echo

FIFO
- RX FIFO Depth
- RX, TX Threshold

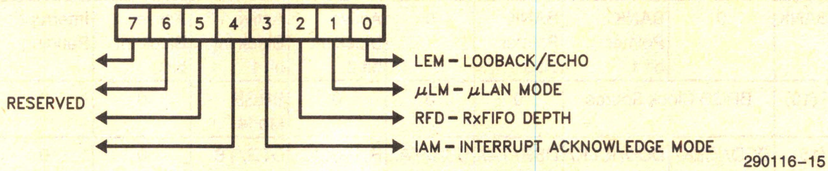
Clock Options
- RX, TX Clock Mode
  - 1X
  - 16X
- RX, TX Clock Source
  - BRGA
  - BRGB
- BRGA/B Operation Mode
  - Timer
  - BRG
- BRGA/B Divide Count
- BRGA/B Source
  - Sys Clock
  - SCLK Pin
  - BRGA Output (BRGB Only)

Control Character Recognition
- None
- Standard
  - ASCII
  - EBCDIC
- Two User Programmed

TX Operation
- RTS/CTS Control
  Manual, Semi-Automatic, Automatic
- Parity Mode
- Stop Bit Length
- Character Size

I/O Pins
- Select Function for Each Multifunction Pin
- Select Direction for Multifunction Pin (If Applicable)

1. IMD—INTERNAL MODE REGISTER



IMD—Internal Mode Register

This register defines the general device mode of operation. The bit functions are as follows:

7–4: Reserved

IAM: Interrupt Acknowledge Mode Bit

0 — Manual acknowledgement of pending interrupts

1 — Automatic acknowledgement of pending interrupts (upon CPU service)

This bit, when set, configures the 82510 for the automatic acknowledge mode. This causes the 82510 INT line to go low for two clock cycles upon service of the interrupt. After two clock cycles it is then updated. It is useful in the edge triggered mode. In manual acknowledgement mode the CPU must explicitly issue a command to clear the INT pin. (The INT pin then goes low for a minimum of two clock cycles until another enabled status register bit is set.)

RFD: Receive FIFO Depth

0 — Four Bytes
1 — One Byte

This bit configures the depth of the Rx FIFO. With a FIFO depth of one, the FIFO will act as a 1-byte buffer to emulate the 8250A.

ULM: uLAN Mode

0 — Normal Mode
1 — uLAN Mode

This bit, enables the 82510 to recognize and/or match address using the 9-bit MCS-51 asynchronous protocol.

LEM: Loopback/Echo Mode Select

This bit selects the mode of loopback operation, or the mode of echo operation; depending upon which operation mode is selected by the Modem Control register bit LC.

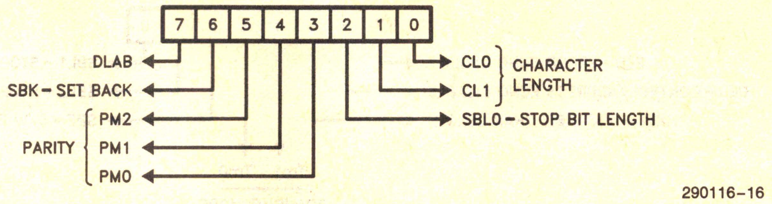
In *loopback* mode (Modem Control register bit LC = 1) it selects between local and remote loopback.

0 — Local Loopback
1 — Remote Loopback

In *echo* mode (Modem Control register bit LC = 0) it selects between echo or non-echo operation.

0 — No Echo
1 — Echo Operation

2. LCR—LINE CONFIGURE REGISTER



LCR—Line Configure Register

This register defines the basic configuration of the serial link.

DLAB—Divisor Latch Access Bit—This bit, when set, allows access to the Divisor Count registers BAL,BAH;BBL,BBH registers.

SBK—Set Break Bit—This bit will force the TxD pin low. The TxD pin will remain low (regardless of all activities) until this bit is reset.

PM2—PM0—Parity Mode Bits—These three bits combine with the SPF bit of the Transmit Mode register to define the various parity modes. See Table 12.

Table 12. Parity Modes

PM0	SPF	PM2	PM1	Function
0	X	X	X	No Parity
1	0	0	0	Odd Parity
1	0	0	1	Even Parity
1	0	1	0	High Parity
1	0	1	1	Low Parity
1	1	0	0	Software Parity

SBL0—Stop Bit Length—This bit, together with SBL1 and SBL2 bits of the Transmit Mode register, defines the Stop-bit lengths for transmission. The Rx machine can identify 3/4 stop bit or more. See Table 13.

Table 13. Stop Bit Length

SBL2	SBL1	SBL0	Stop Bit Length 16X	1X
0	0	0	4/4	—
0	0	1	6/4 or 8/4*	—
0	1	0	3/4	1
0	1	1	4/4	1
1	0	0	5/4	1
1	0	1	6/4	1
1	1	0	7/4	1
1	1	1	8/4	2

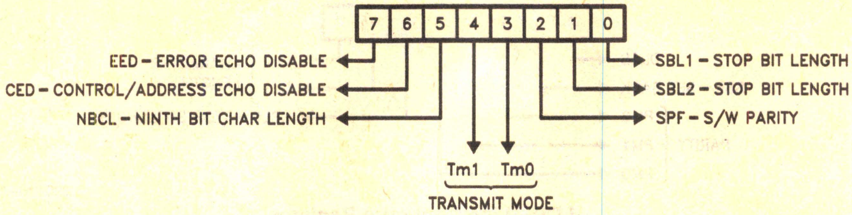
*6/4 if character length is 5 bits; else 8/4

CL0—CL1—Character Length Bits—These bits, together with the Transmit Mode register bit NBCL, define the character length. See Table 14.

Table 14. Character Length

NBCL	CL1	CL0	Character Length
0	0	0	5 BITS
0	0	1	6 BITS
0	1	0	7 BITS
0	1	1	8 BITS
1	0	0	9 BITS

3. TMD—Transmit Machine Mode Register



290116-17

TMD—Transmit Machine Mode Register

This register together with the Line Configure Register defines the Tx machine mode of operation.

EED—Error Echo Disable—Disables Echo of characters received with errors (valid in echo mode only).

CED—Control Character Echo Disable—Disables Echo of characters recognized as control characters (or address characters in uLAN mode). Valid in echo mode only.

NBCL—Nine-Bit Length—This bit, coupled with LCR (CL0, CL1), selects Transmit/Receive character length of nine bits. See Table 14.

TM1—TM0—Transmit Mode—These bits select one of three modes of control over the $\overline{CTS}$ and $\overline{RTS}$ lines.

00—Manual Mode—In this mode the CPU has full control of the Transmit operation. The CPU has to explicitly enable/disable transmission, and activate/check the $\overline{RTS}/\overline{CTS}$ pins.

01—Reserved

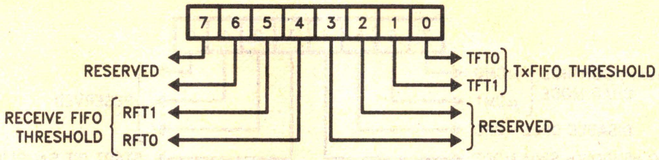
10—Semi-Automatic Mode—In this mode the 82510 transmits only when $\overline{CTS}$ input is active. The 82510 activates the $\overline{RTS}$ output as long as transmission is enabled.

11—Automatic Mode—In this mode the 82510 transmits only when $\overline{CTS}$ input is active. The $\overline{RTS}$ output is activated only when transmission is enabled and there is more data to transmit.

SPF—Software Parity Force—This bit defines the parity modes along with the PM0, PM1, and PM2 bits of the LCR register. When software parity is enabled the software must determine the parity bit through the TxF register on transmission, or check the parity bit in RxF upon reception. See Table 12.

SBL2—SBL1—Stop Bit Length—These bits, together with the SBL0 bit of the LCR register define the stop bit length. See Table 13.

4. FMD—FIFO MODE REGISTER



290116-18

FMD—FIFO Mode Register

This register configures the Tx and Rx FIFO's threshold levels—the occupancy levels that can cause an interrupt.

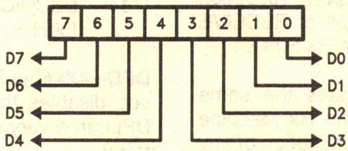
7—6—Reserved

RFT1—RFT0—Receive FIFO Threshold—When the Rx FIFO occupancy is greater than the level indicated by these bits the Rx FIFO Interrupt is activated.

3—2—Reserved

TFT1—TFT0—Transmit FIFO Threshold—When the TX FIFO occupancy is less than or equal to the level indicated by these bits the Tx FIFO Interrupt is activated.

5. ACR0—ADDRESS/CONTROL CHARACTER REGISTER 0



290116-47

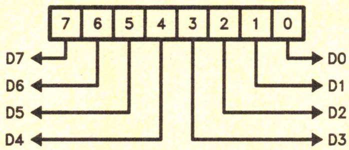
ARC0—Address/Control Character Register 0

This register contains a byte which is compared to each received character. The exact function depends on the configuration of the IMD register.

In normal mode this register may be used to program a special control character; a matched character will be reported in the RECEIVE STATUS register. The maximum length of the control characters is eight bits. If the length is less than eight bits then the character must be right justified and the leading bits be filled with zeros.

In uLAN mode this register contains the eight-bit station address for recognition. In this mode only incoming address characters (i.e., characters with address bit set) will be compared to these register. The PCRF bit in the RECEIVE STATUS register will be set when an Address or Control Character match occurs.

6. ACR1—ADDRESS/CONTROL CHARACTER REGISTER 1

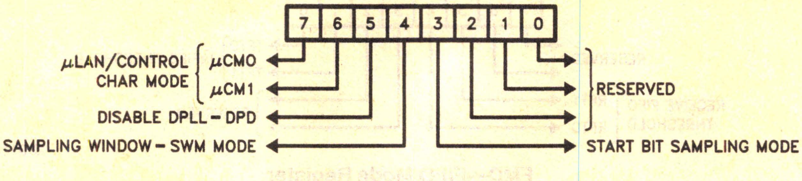


290116-48

ARC1—Address/Control Character Register 1

NOTE:
This register is identical in function to ACR0.

7. RMD—RECEIVE MACHINE MODE REGISTER



RMD—Receive Machine Mode Register

This register defines the Rx Machine mode of operation.

uCM0, uCM1—uLAN/Control Character Recognition Mode—In normal mode it defines the Control Character recognition mode. In ulan mode they define modes of address recognition.

In *uLAN* mode: selects the mode of address recognition.

00—Manual Mode—Rx Machine reports reception of any address character, via CRF bit of RECEIVE STATUS register, and writes it to the Rx FIFO.

01—Semi-Automatic Mode—Operates the same as Manual Mode but, in addition, the Rx Machine OPENS (unlocks) the Rx FIFO upon reception of any address characters. Subsequent received characters will be written into the FIFO. (Note: it is the user's responsibility to LOCK the FIFO if the address character does not match the station's address.)

10—Automatic Mode—The Rx Machine will OPEN (unlock) the Rx FIFO upon Address Match. In addition the Rx Machine LOCKs the Rx FIFO upon recognition of address mismatch; i.e., it controls the flow of characters into the Rx FIFO depending upon the results of the address comparison. If a match occurs it will allow characters to be sent to the FIFO; if a mismatch occurs it will keep the characters out of the FIFO by LOCKING it.

11—Reserved

In *normal* Mode: selects the mode of Standard Set Control Character Recognition (programmed control characters are always recognized).

00— No Standard Set Control Characters Recognized.

01— ASCII Control Characters (00H—1 FH + 7FH).

10— Reserved.

11— EBCDIC Control Character Recognized (00H – 3FH).

DPD—Disable Digital Phase Locked Loop—When set, disables the DPLL machine. (Note: using the DPLL in a very noisy media, may increase the error rate.)

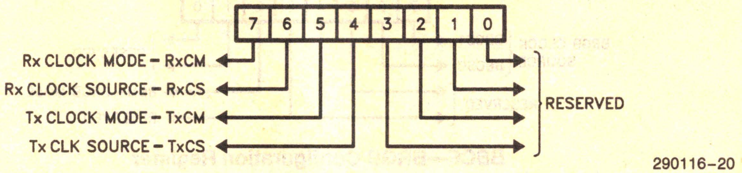
SWM—Sampling Window Mode—This bit controls the mode of data sampling:

0—Small Window, 3/16 sampling.
1—Large Window, 7/16 sampling.

SSM—Start Bit Sampling Mode—This bit controls the mode of Start Bit sampling.

0— Majority Voting for start bit. In this mode a majority of the samples determines the bit.
1— In this mode if one of the bit samples is not '0', the start bit will not be detected.

8. CLCF—CLOCKS CONFIGURE REGISTER



CLCF—Clocks Configure Register

This register defines the Transmit and Receive Code modes and sources.

RxCM—Rx Clock Mode—This bit selects the mode of the receive clock which is used to sample the received data.

0— 16X Mode.

1— 1X Mode. In this mode the receive data must be synchronous to the Rx Clock; supplied via the SCLK pin.

RxCS—Rx Clock Source—This bit selects the source of the internal receive clock in the case of 16X mode (as programmed by the RxCM bit above).

0—BRGB Output
1—BRGA Output

TxCM—Transmit Clock Mode—This bit selects the mode of the Transmit Data Clock, which is used to clock out the Transmit Data.

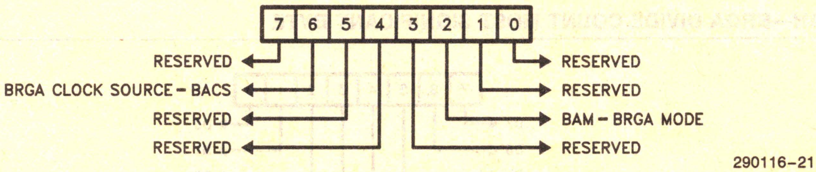
0— 16X Mode

1— 1X Mode. In this mode the Transmit data is synchronous to the Serial Clock; supplied via the SCLK pin.

TxCS—Transmit Clock Source—Selects the source of internal Transmit Clock in case of 16X mode.

0—BRGB Output.
1—BRGA Output.

9. BACF—BRGA CONFIGURATION REGISTER



BACF—BRGA Configuration Register

This register defines the BRGA clock sources and the mode of operation.

BACS—BRGA Clock Source—Selects the input clock source for Baud Rate Generator A.

0—System Clock
1—SCLK Pin

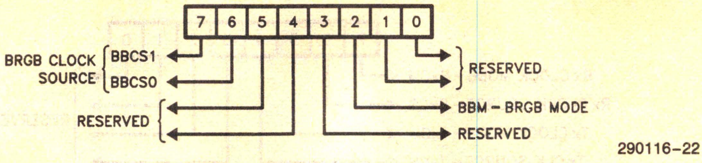
This bit has no effect if BRGA is configured as a timer.

BAM—BRGA Mode of Operation—Selects between the Timer mode or the Baud Rate Generator Mode.

0— Timer Mode (in this mode the input clock source is always the system clock).

1— Baud Rate Generator Mode

10. BBCF—BRGB CONFIGURATION REGISTER



BBCF—BRGB Configuration Register

This register defines the BRGB clock sources and mode of operation. (Note: BRGB can also take its Input Clock from the output of BRGA.)

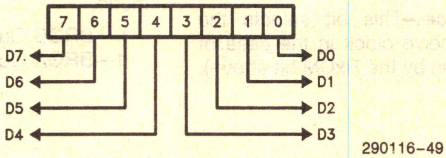
BBCS1, BBCS0—These two bits together define the input Clock Sources for BRGB. These bits have no effect when in the timer mode.

- 00— System Clock
- 01— SCLK Pin
- 10— BRGA Output
- 11— Reserved

BBM—BRGB Mode of Operation.

- 0— Timer Mode (In this mode the input clock source is always the system clock.)
- 1— BRG Mode

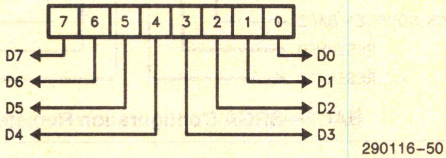
11. BAL—BRGA DIVIDE COUNT LEAST SIGNIFICANT BYTE



BAL—BRGA Divide Count Low Byte

This register contains the least significant byte of the BRGA divisor/count.

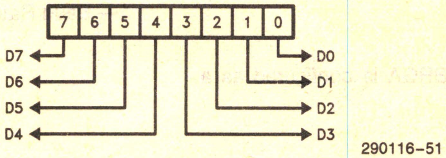
12. BAH—BRGA DIVIDE COUNT MOST SIGNIFICANT BYTE



BAH—BRGA Divide Count High Byte

This register contains the most significant byte of the BRGA divisor/count.

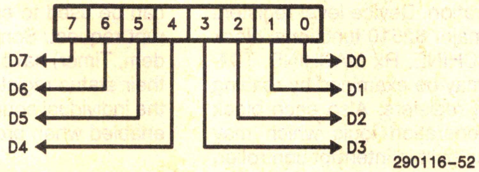
13. BBL—BRGB DIVIDE COUNT LEAST SIGNIFICANT BYTE



BBL—BRGB Divide Count Low Byte

This register contains the least significant byte of the BRGB divisor/count.

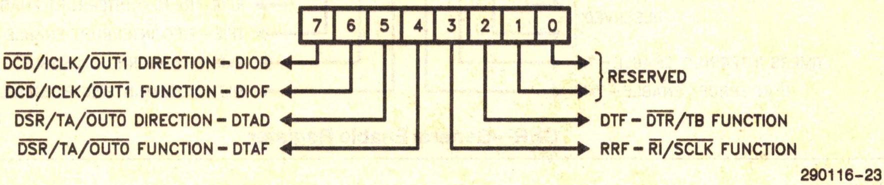
14. BBH—BRGB DIVIDE COUNT MOST SIGNIFICANT BYTE



BBH—BRGB Divide Count High Byte

This register contains the most significant byte of the BRGB divisor/count.

15. PMD—I/O PIN MODE REGISTER



PMD—I/O Pin Mode Register

This register is used to configure the direction and function of the multifunction pins. The following options are available on each pin.

1. Direction: Input or Output Pin.
   - 0— Defines the Pin as an output pin (general purpose or special function).
   - 1— Defines the pin as an input pin.
2. Function: General purpose or special purpose pin (no effect if the pin is programmed as an Input).
   - 0— special function output pin.
   - 1— general purpose output pin.

DIOD—$\overline{\text{DCD}}$/ICLK/$\overline{\text{OUT1}}$ Direction.

- 0— Output: ICLK or $\overline{\text{OUT1}}$ (depending on bit DIOF)
- 1— Input: $\overline{\text{DCD}}$.

DIOF—$\overline{\text{DCD}}$/ICLK/$\overline{\text{OUT1}}$ Function (output mode only).

- 0— ICLK (Output of the Internal System Clock).
- 1— $\overline{\text{OUT1}}$ general purpose output, Controlled by MODEM CONTROL Register

DTAD—$\overline{\text{DSR}}$/TA/$\overline{\text{OUT0}}$ Direction.

- 0— Output: TA or $\overline{\text{OUT0}}$ (Dependent upon DTAF).
- 1— Input: $\overline{\text{DSR}}$.

DTAF—$\overline{\text{DSR}}$/TA/$\overline{\text{OUT0}}$ Direction (output mode only).

- 0— TA (BRGA Output or Timer A Termination Pulse).
- 1— $\overline{\text{OUT0}}$ (general purpose output, controlled by MODEM CONTROL).

RRF—$\overline{\text{RI}}$/SCLK Function

- 0— SCLK (Receive and/or Transmit Clock)
- 1— $\overline{\text{RI}}$

DTF—$\overline{\text{DTR}}$/TB Function

- 0— TB (BRGB Output Clock on Timer B termination pulse depending upon the mode of BRGB).
- 1— $\overline{\text{DTR}}$

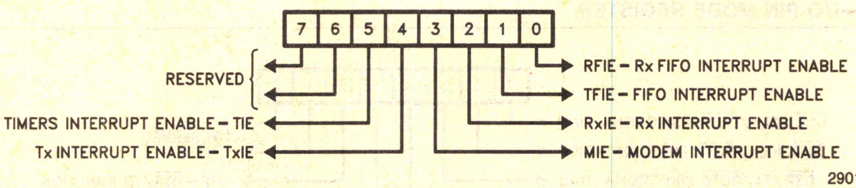
INTERRUPT/STATUS REGISTERS

The 82510 uses a two layer approach to handle interrupt and status generation. Device level registers show the status of the major 82510 functional block (MODEM, FIFO, Tx MACHINE, Rx MACHINE, TIMERS, etc.). Each block may be examined by reading its individual block level registers. Also each block has interrupt enable/generation logic which may generate a request to the built-in interrupt controller, the interrupt requests are then resolved on a priority basis.

Interrupt Masking

The 82510 has a device enable register, GER, which can be used to enable or mask-out any block interrupt request. Some of the blocks (Rx Machine, Modem, Timer) have an enable register associated with their status register which can be used to mask out the individual sources within the block. Interrupts are enabled when programmed high.

16. GER—GENERAL ENABLE REGISTER



GER—General Enable Register

This register enables or disables the bits of the GSR register from being reflected in the GIR register. It serves as the device enable register and is used to mask the interrupt requests from any of the 82510 block (See Figure 1).

TIE—Timers Interrupt Enable

TxIE—Transmit Machine Interrupt Enable.

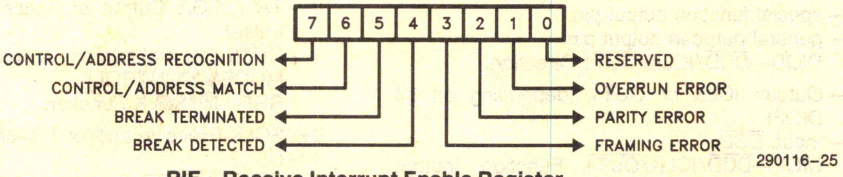
MIE—Modem Interrupt Enable.

RxIE—Rx Machine Interrupt Enable.

TFIE—Transmit FIFO Interrupt Enable.

RFIE—Receive FIFO Interrupt Enable.

17. RIE—RECEIVE INTERRUPT ENABLE REGISTER



RIE—Receive Interrupt Enable Register

This register enables interrupts from the Rx Machine. It is used to mask out interrupt requests generated by the status bits of the RST register.

CRE—Control/uLAN Address Character Recognition Interrupt Enable.—Enables Interrupt when CRF bit of RST register is set.

PCRE—Programmable Control/Address Character Match Interrupt Enable.—Enables Interrupt on PCRF bit of RST.

BkTe—Break Termination Interrupt Enable.

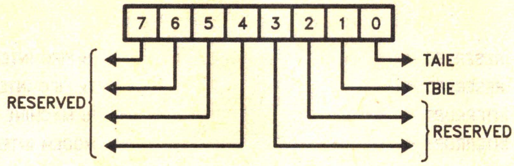
BkDE—Break Detection Interrupt Enable— Enable Interrupt on BkD bit of RST.

FEE—Framing Error Enable—Enable Interrupt on FE bit of RST.

PEE—Parity Error Enable—Enable Interrupt on PE bit of RST.

OEE—Overrun Error Enable—Enable Interrupt on OE bit of RST.

18. TMIE—TIMER INTERRUPT ENABLE REGISTER



290116-26

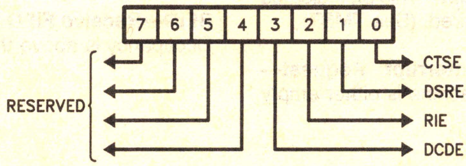
TMIE—Timers/Interrupt Enable Register

This is the enable register for the Timer Block. It is used to mask out interrupt requests generated by the status bits of the TMST register.

TBIE—Timer B Expired Interrupt Enable— Enables Interrupt on TBEx bit of TMST.

TAIE—Timer A Expired Interrupt Enable— Enables Interrupt on TAEx bit of TMST.

19. MIE—MODEM INTERRUPT ENABLE REGISTER



290116-27

MIE—Modem Interrupt Enable Register

This register enables interrupts from the Modem Block. It is used to mask out interrupt requests generated by the status bits of the MODEM STATUS register.

DCDE—Delta $\overline{\text{DCD}}$ Interrupt Enable—Enables Interrupt on DDCD bit of MODEM STATUS.

RIE—Delta $\overline{\text{RI}}$ Interrupt Enable—Enables Interrupt on DRI bit of MODEM STATUS.

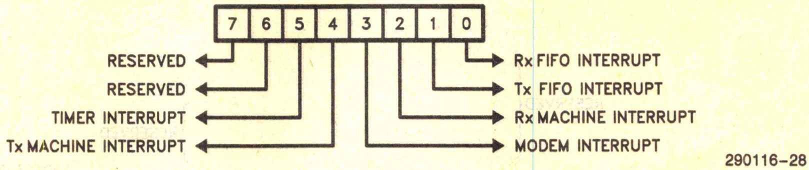
DSRE—Delta $\overline{\text{DSR}}$ Interrupt Enable—Enables Interrupt on $\overline{\text{DSR}}$ bit of MODEM STATUS.

CTSE—Delta $\overline{\text{CTS}}$ Interrupt Enable—Enables Interrupt on DCTS bits of MODEM STATUS.

STATUS/INTERRUPT

The 82510 has two device status registers, which reflect the overall status of the device, and five block status registers. The two device status registers, GSR and GIR, and supplementary in function. GSR reflects the interrupt status of all blocks, whereas GIR depicts the highest priority interrupt only. GIR is updated after the GSR register; the delay is of approximately two clock cycles.

20. GSR—GENERAL STATUS REGISTER



GSR—General Status Register

This register reflects all the pending block-level Interrupt requests. Each bit in GSR reflects the status of a block and may be individually enabled by GER. GER masks-out interrupts from GIR; it does not have any effect on the bits in GSR. The only way that the bits can be masked out in GSR (i.e., not appear in GSR) is if they are masked out at the lower level.

TIR—Timers Interrupt Request—This bit indicates that one of the timers has expired. (See TMST)

TxIR—Transmit Machine Interrupt Request—Indicates that the Transmit Machine is either empty or disabled (Idle).

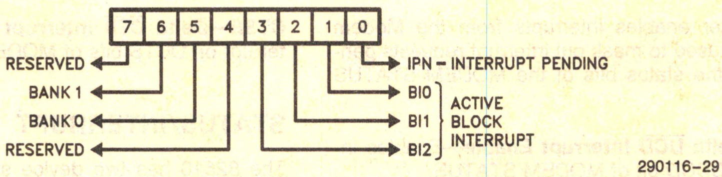
MIR—Modem Interrupt Request—This bit, if set, indicates an interrupt from the Modem Module. (As reflected in MODEM STATUS.)

RxIR—Receive Machine Interrupt Request—(As reflected in RST.)

TFIR—Transmit FIFO Interrupt Request—Tx FIFO occupancy is below or equal to threshold.

RFIR—Receive FIFO Interrupt Request—Rx FIFO Occupancy is above threshold.

21. GIR/BANK—GENERAL INTERRUPT REGISTER/BANK REGISTER



General Interrupt Register/Bank Register

This register holds the highest priority enabled pending interrupt from GSR. In addition it holds a pointer to the current register segment. Writing into this register will update only the BANK bits.

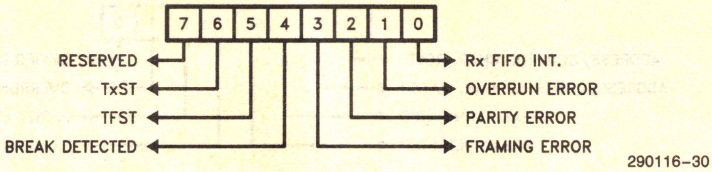
BANK1, BANK0—Bank Pointer Bits—These two bits point to the currently accessible register bank. The user can read and write to these bits. The address of this register is always two within all four register banks.

BI2, BI1, BI0,—Interrupt Bits 0–2—These three bits reflect the highest priority enabled pending interrupt from GSR.

101: Timer Interrupt (highest priority)
100: Tx Machine Interrupt
011: Rx Machine Interrupt
010: Rx FIFO Interrupt
001: Tx FIFO Interrupt
000: Modem Interrupt (lowest priority)

IPN—Interrupt Pending—This bit is active low. It indicates that there is an interrupt pending. The interrupt logic asserts the INT pin as soon as this bit goes active. (Note: the GIR register is continuously updated; so that, while the user is serving one interrupt source, a new interrupt with higher priority may enter GIR and replace the older interrupt.)

22. LSR—LINE STATUS REGISTER



LSR—Line Status Register

This register holds the status of the serial link. It shares five of its bits with the RST register (BkD, FE, PE, OE, and RFIR). When this register is read, the RST register (BITS 1–7) and LSR register (BITS 1–4) are cleared. This register is provided for compatibility with the INS8250A.

TxSt—Transmit Machine Status Bit—Same as TxIR bit of GSR register. If high it indicates that the Transmit Machine is in Idle State. (Note: Idle may indicate that the TxM is either empty or disabled.

TFSt—Transmit FIFO Status—Same as TFIR bit of GSR. It indicates that the Transmit FIFO level is equal to or below the Transmit FIFO Threshold. There are two ways to disable the transmit FIFO status from being reflected in GIR:

1. Writing a "0" to the TFIE bit of the GER register
2. Dynamically by using the Tx FIFO HOLD INTERRUPT logic. When the Tx FIFO is in the Hold State, no interrupts are generated regardless of the TFIR and TFIE bits.

The Transmit FIFO enters the Hold State when the CPU reads the GIR register and the source of the interrupt is Tx FIFO. To Exit, the CPU must drop the TFIR bit of GSR by writing a character to Tx FIFO, or drop TFIE bit of GER (Disable Tx FIFO).

Bkd—Break Detected—See Bkd bit in RST register for full explanation. The BkD bit in RST register is the same as this bit.

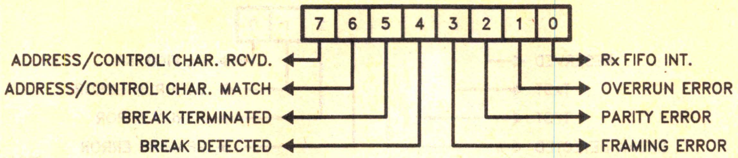
FE—Framing Error Detected—See FE bit in RST register for a full explanation. The FE bit in RST register is the same as this bit.

PE—Parity Error Detected—See PE bit in RST register for full explanation. The PE bit in RST register is the same as this bit.

OE—Overrun Error—See OE bit in RST register for full explanation. The OE bit in RST register is the same as this bit.

RFIR—Receive FIFO Interrupt Request—This bit is identical to RFIR bit of GSR. It indicates that the RX FIFO level is above the Rx FIFO Threshold. This bit is forced LOW during any READ from the Rx FIFO. A zero written to this bit will acknowledge an Rx FIFO interrupt.

23. RST—RECEIVE MACHINE STATUS REGISTER



290116-31

RST—Receive Machine Status Register

This register displays the status of the Receive Machine. It reports events that have occurred since the RST was cleared. This register is cleared when it is read except for BIT0, Rx FIFO interrupt. Each bit in this register, when set, can cause an interrupt. Five bits of this register are shared with the LSR register.

CRF—Control/Address Character Received— When enabled, this bit can cause an interrupt if a control character or address character is received.

In uLAN Mode: indicates that an address character has been received.
In normal Mode: indicates that a standard control character (either ASCII or EBCDIC) has been received.

PCRF—Programmed Control/Address Character Received—This bit, when enabled, will cause an interrupt when an address or control character match occurs.

In uLAN Mode: indicates that an address character equal to one of the registers ACR0 or ACR1 has been received.

In normal Mode: indicates that a character which matches the registers ACR0 or ACR1 has been received.

BkT—Break Terminated—This bit indicates that a break condition has been terminated.

BkD—Break Detected—This bit indicates that a Break Condition has been detected, i.e., RxD input was held low for one character frame plus a stop BIT.

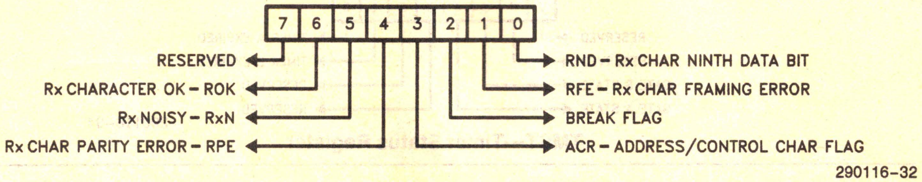
FE—Framing Error—This bit indicates that a received character did not have a valid stop bit.

PE—Parity Error—Indicates that a received character had a parity error.

OE—Overrun Error—Indicates that a received character was lost because the Rx FIFO was full.

RFIR—Receive FIFO Interrupt Request—Same as the RFIR bit of LSR register.

24. RXF—RECEIVED CHARACTER FLAGS



RxF—Receive Flags Register

This register contains additional information about the character in the RXD register. It is loaded by the Rx Machine simultaneously with the RXD register.

ROK—Received Character OK—This bit indicates that the character in RXD no parity or framing error. The parity error is not included in the s/w parity mode.

RxN—Received Character Noisy—The character in RXD was noisy. This bit, valid only in 16X sampling mode, indicates that the received character had non-identical samples for at least one of its bits.

RPE—Receive Character Parity Error—This bit indicates that the RxD character had a Parity Error. However, in S/W Parity mode it holds the received parity bit as is.

ACR—Address/Control Character Marker—This bit indicates that the character in RXD is either:

A control Character—in normal Mode.
An Address Character in uLAN Mode.

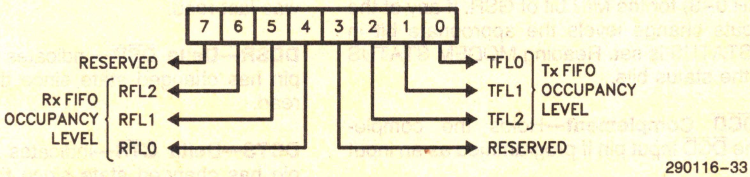
RFE—Receive Character Framing Error—Indicates that no Stop bit was found for the character in RXD.

NOTE:
A Framing Error will be generated for the first character of the Break sequence.

RND—Ninth Bit of Received Character—The most significant bit of the character in RXD is written into this bit. This bit is zero for characters with less than nine bits.

BKF—Break Flag—Indicates that the character is part of a "break" sequence.

25. FLR—FIFO LEVEL REGISTER



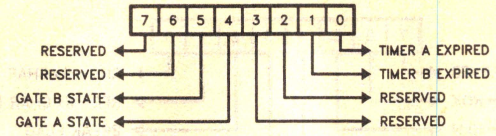
FLR—FIFO Level Register

This register holds the current Receive and Transmit FIFO occupancy levels.

RFL2, RFL1, RFL0—Receive FIFO Level of Occupancy—These three bits indicate the level of Occupancy of the Rx FIFO. The valid range is zero (000) to four (100).

TFL2, TFL1, TFL0—Transmit FIFO Level of Occupancy—These three bits indicate the level of occupancy in the transmit FIFO. The valid range is zero (000) to four (100).

26. TMST—TIMER STATUS REGISTER



290116-34

TMST—Timer Status Register

This register holds the status of the timers. Bits TBEx and TAEx generate interrupts which are reflected in bit TIR of GSR. Bits GBS and GAS just display the counting status, they do not generate interrupts.

GBS—Gate B State—This bit does not generate an interrupt. It indicates the counting state of the software gate of Timer B, as written through the TMCR register.

0—counting disabled
1—counting enabled

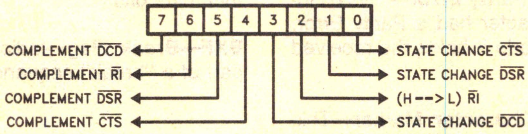
GAS—Gate—A State—This bit does not generate an interrupt. It reflects the state of the software gate of Timer A, as written through the TMCR register.

0—counting disabled
1—counting enabled

TBEx—Timer B Expired—When Set generates an interrupt through TIR bit of GSR. Indicates that Timer B count has expired. This bit is set via the terminal count pulse generated by the timer when it terminates counting.

TAEx—Timer A Expired—Same as TBEx except it refers to Timer A.

27. MSR—MODEM/I/O PINS REGISTER



290116-35

MSR—Modem/I/O Pins Status Register

This register holds the status of the Modem input pins (CTS, DCD, DSR, RI). It is the source of interrupts (MSR 0-3) for the MIR bit of GSR. If any of the above inputs change levels the appropriate bit in MODEM STATUS is set. Reading MODEM STATUS will clear the status bits.

DCDC—$\overline{DCD}$ Complement—Holds the complement of the $\overline{DCD}$ input pin if programmed as an input in PMD.

DRIC—Holds the complement of the $\overline{RI}$ input pin if programmed as an input in PMD.

DSRC—$\overline{DSR}$ Complement—Holds the complement of the $\overline{DSR}$ input pin if configured as an input in PMD.

CTSC—$\overline{CTS}$ Complement—Holds the complement of the $\overline{CTS}$ pin.

DDCD—Delta $\overline{DCD}$—Indicates that the $\overline{DCD}$ input pin has changed state since this register was last read.

DRI—Delta $\overline{RI}$—Indicates that there was a high-to-low transition on the $\overline{RI}$ input pin since the register was last read.

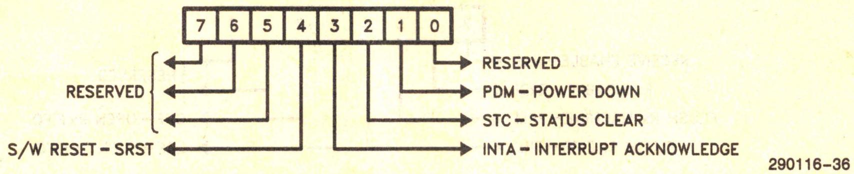
DDSR—Delta $\overline{DSR}$—Indicates that the $\overline{DSR}$ input pin has changed state since this register was last read.

DCTS—Delta $\overline{CTS}$—Indicates that the $\overline{CTS}$ input pin has changed state since this register was last read.

COMMAND REGISTERS

The command registers are write only; they are used to trigger an operation by the device. Once the operation is started the register is automatically reset. There is a device level register as well as four block command registers. It is recommended that only one command be issued during a write cycle.

28. ICM—INTERNAL COMMAND REGISTER



ICM—Internal Command Register

This register activates the device's general functions.

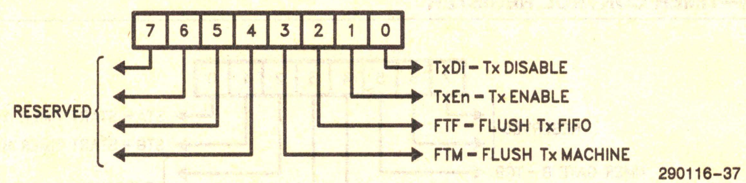
SRST—Device Software RESET—Causes a total device reset; the effect is identical to the hardware reset (except for strapping options). The reset lasts four clocks and puts the device into the Default Wake-up Mode.

INTA—Interrupt Acknowledge—This command is an explicit acknowledgement of the 82510's interrupt request. It forces the INT pin inactive for two clocks; afterwards, the INT pin may again go active if other enabled interrupts are pending. This command is provided for the Manual Acknowledge mode of the 82510.

StC—Status Clear—Clears the following status registers: RST, MSR, and TMST.

PDM—Power Down—This command forces the device into the power-down mode. Refer to the functional description for details.

29. TCM—TRANSMIT COMMAND REGISTER



TCM—Transmit Command Register

This register controls the operation of the Transmit Machine.

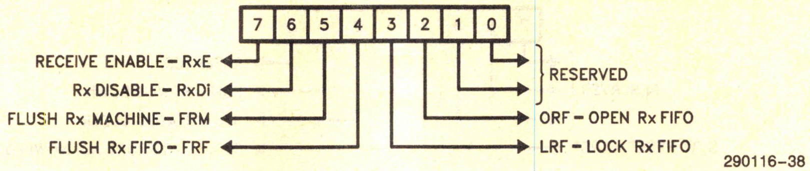
FTM—Flush Transmit Machine—Resets the Transmit Machine logic (but not the registers or FIFO) and enables transmission.

FTF—Flush Transmit FIFO—Clears the Tx FIFO.

TxEN—Transmit Enable—Enables Transmission by the Transmit Machine.

TxDi—Transmit Disable—Disables transmission. If transmission is occurring when this command is issued the Tx Machine will complete transmission of the current character before disabling transmission.

30. RCM—RECEIVE COMMAND REGISTER



RCM—Receive Command Register

This register controls the operation of the Rx machine.

RxE—Receive Enable—Enables the reception of characters.

RxDi—Receive Disable—Disables reception of data on RXD pin.

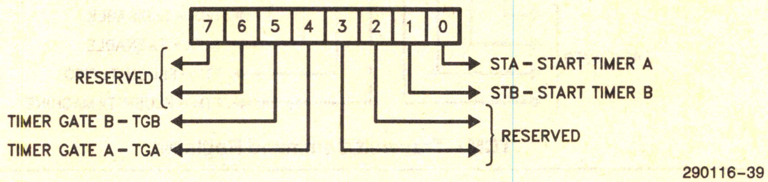
FRM—Flush Receive Machine—Resets the Rx Machine logic (but not registers and FIFOs), enables reception, and unlocks the receive FIFO.

FRF—Flush Receive FIFO—Clears the Rx FIFO.

LRF—Locks Rx FIFO—Disables the write mechanism of the Rx FIFO so that characters subsequently received are not written to the Rx FIFO but are lost. However, reception is not disabled and complete status/event reporting continues. (This command may be used in the uLAN mode to disable loading of characters into the Rx FIFO until an address match is detected.)

ORF—Open (Unlock) Rx FIFO—This command enables or unlocks the write mechanism of the Rx FIFO.

31. TMCR—TIMER CONTROL REGISTER



TMCR—Timer Control Register

This register controls the operation of the two 82510 timers. It has no effect when the timers are configured as baud—rate generators. TGA and TGB are not reset after command execution.

TGB—Timer-B Gate—This bit serves as a gate for Timer B operation:

1—enables counting
0—disables counting

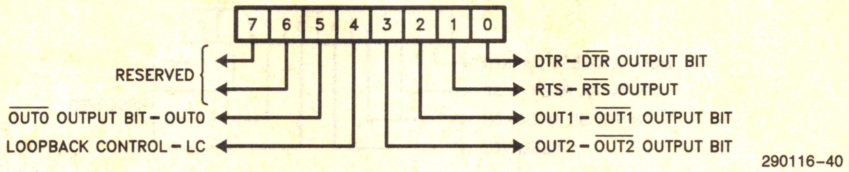
TGA—Timer-A Gate—This bit serves as a gate for Timer-A operation:

1—enables counting
0—disables counting

STB—Start Timer B—This command triggers timer B. At terminal count a status bit is set in TMST (TBEx).

STA—Start Timer A—This command triggers timer A. At terminal count a status bit is set in TMST (TAEx).

32. MCR—MODEM CONTROL REGISTER



MCR—Modem Control Register

This register controls the modem output pins. With multi—function pins it affects only the pins configured as general purpose output pins. All the output pins invert the data, i.e. their output will be the complement of the data written into this register.

OUT0—$\overline{\text{OUT0}}$ Output Bit—This bit controls the $\overline{\text{OUT0}}$ pin. The output signal is the complement of this bit.

LCB Loopback Control Bit—This bit puts the 82510 into loopback mode. The particular type of loopback is selected via the IMD register.

OUT2—$\overline{\text{OUT2}}$ Output Bit—This bit controls the $\overline{\text{OUT2}}$ pin. The output signal is the complement of this bit.

OUT1—$\overline{\text{OUT1}}$ Output Bit—This bit controls the $\overline{\text{OUT1}}$ pin. The output signal is the complement of this bit.

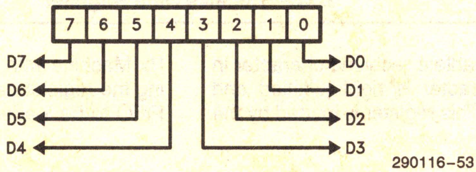
RTS—$\overline{\text{RTS}}$ Output Bit—This bit controls the $\overline{\text{RTS}}$ pin. The output signal is the complement of this bit.

DTR—$\overline{\text{DTR}}$ Output Bit—This bit controls the $\overline{\text{DTR}}$ pin. The output signal is the complement of this bit.

DATA REGISTERS

The data registers hold data or other information and may be accessed at any time.

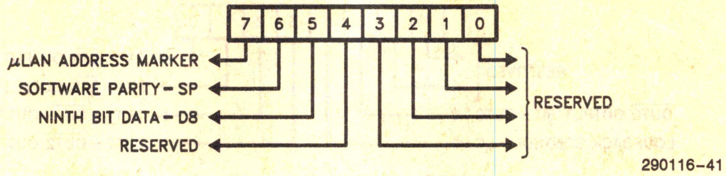
33. TXD—TRANSMIT DATA REGISTER



TXD—Transmit Data Register

This register holds the next data byte to be pushed into the Transmit FIFO. For character formats with more than eight bits of data, or with additional components (S/W Parity, Address Marker Bit) the additional data bits should be written into the TxF register. When a byte is written to this register its contents, along with the contents of the TxF register, are pushed to the top of the Transmit FIFO. This register is write only.

34. TXF—TRANSMIT FLAGS REGISTER



TxF—Transmit Flags Register

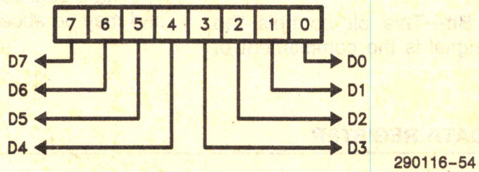
This register holds some additional components of the next character to be pushed into the Tx FIFO. The contents of this register are pushed into the Tx FIFO with the Transmit Data register whenever the TxD register is written to by the CPU.

uLAN—uLAN Address Marker Bit—This bit is transmitted in uLAN mode as the address marker bit.

SP—Software Parity Bit—This bit is transmitted in S/W parity mode as the character's parity bit.

D8—Ninth Bit of Data—In nine-bit character length mode this bit is transmitted as the MSB (D8) bit.

35. RXD—RECEIVE DATA REGISTER



RXD—Receive Data Register

This register holds the earliest received character in the Rx FIFO. The character is right justified and leading bits are zeroed. This register is loaded by the Rx Machine with the first received character. Reading the register causes the next register from the Rx FIFO to be loaded into RxD and RxF registers.

SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature under Bias ......0°C to 70°C
Storage Temperature .... −65° to +150°C
Voltage on any Pin (w.r.t. V_{SS}) −0.5V to V_{CC} +0.5V
Voltage on V_{CC} Pin (w.r.t. V_{SS})...... −0.5V to +7V
Power Dissipation ....300 mW

D.C. SPECIFICATIONS

D.C. CHARACTERISTICS (T_A = 0° to 70°C, V_{CC} = 5V ±10%)

Symbol	Parameter	Notes	Min	Max	Units
V_{IL}	Input Low Voltage	(1)	−0.5	0.8	V
V_{IH}	Input High Voltage	(1)	2.0	V_{CC}−0.5	V
V_{OL}	Output Low Voltage	(2), (9)		0.45	V
V_{OH}	Output High Voltage	(3), (9)	2.4		V
I_{LI}	Input Leakage Current	(4)		±10	μA
I_{LO}	3-State Leakage Current	(5)		±10	μA
I_{cc}	Power Supply Current	(6)		3.8	mA/MHz
I_{pd}	Power Down Supply	(7)		2	mA
I_{STBY}	Standby Supply Current	(10)		TBD	μA
I_{OHR}	$\overline{RTS}$, $\overline{DTR}$ Strapping Current	(11)		0.4	mA
I_{OLR}	$\overline{RTS}$, $\overline{DTR}$ Strapping Current	(12)	11		mA
C_{in}	Input Capacitance	(8)	10		pF
C_{io}	I/O Capacitance	(8)	10		pF
C_{XTAL}	X1, X2 Load			10	pF

NOTES:
1. Does not apply to CLK/X1 pin, when configured as crystal oscillator input (X1).
2. @ I_{OL} = 2 mA.
3. @ I_{OH} = −0.4 mA.
4. 0 < V_{IN} < V_{CC}.
5. 0.45V < V_{OUT} < (V_{CC} − 0.45).
6. V_{CC} = 5.5V; V_{IL} = 0.5V (max); V_{IH} = V_{CC} − 0.5V (min); Typical value = 2.5 mA/MHz (Not Tested); Ext 1X CLK; I_{OL} = I_{OH} = 0.
7. V_{CC} = 5.5V; V_{IL} = GND; V_{IH} = V_{CC}; I_{OL} = I_{OH} = 0; device at power down mode, clock running.
8. Freq = 1 MHz.
9. Does not apply to OUT2/X2 pin, when configured as crystal oscillator output (X2).
10. Same as 7, but input clock not running.
11. Applies only during hardware reset for clock configuration options. Strapping current for logic HIGH.
12. Applies only during hardware reset for clock configuration. Strapping current for logic LOW.

A.C. SPECIFICATIONS

Testing Conditions:

- All AC output parameters are under output load of 20 to 100 pF, unless otherwise specified.
- AC testing inputs are driven at 2.4 for logic '1', and 0.45V for logic '0'. Output timing measurements are made at 1.5V for both a logical '0' and '1'.
- In the following tables, the units are ns, unless otherwise specified.

System Interface Specification—System Clock Specification:

The 82510 system clock is supplied via the CLK pin or generated by an on-chip crystal oscillator. The clock is optionally divided by two. The CLK parameters are given separately for internal divide-by-two option ACTIVE and INACTIVE.

The system clock (after division by two, if active) must be at least 16X the Tx or Rx baud rate (the faster of the two).

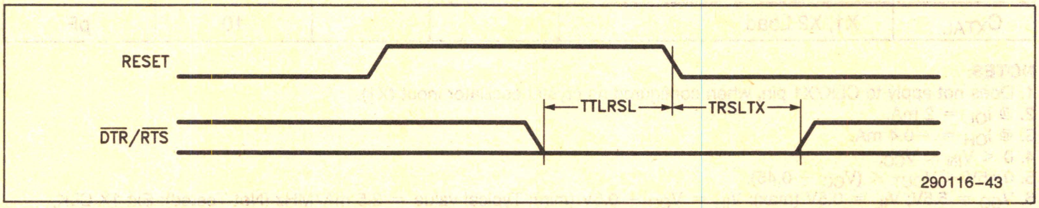
SYSTEM CLOCK SPECIFICATIONS

Symbol	Parameter	Min	Max	Notes
DIVIDE BY TWO OPTION—ACTIVE				
Tcy/2	CLK Period	54	250	(2)
TCLCH	CLK Low Time	25		
TCHCL	CLK High Time	25		
TCH1CH2	CLK Rise Time		10	(1)
TCL2CL1	CLK Fall Time		10	(1)
FXTAL	External Crystal Frequency Rating	4.0	18.432 MHz	
DIVIDE BY TWO OPTION—INACTIVE				
Tcy	CLK Period	108		
TCLCH	CLK Low Tme	54		
TCHCL	CLK High Time	44	250	
TCH1CH2	CLK Rise Time		15	(1)
TCL2CL1	CLK Fall Time		15	(1)

NOTES:
1. Rise/fall times are measured between 0.8 and 2.0V.
2. Tcy in ACTIVE divide by two option is TWICE the input clock period.

RESET SPECIFICATION

Symbol	Parameter	Min	Max	Notes
TRSHL	Reset Width—CLK/X1 Configured to CLK	8 Tcy		(1)
TTLRSL	RTS/DTR LOW Setup to Reset Inactive	6 Tcy		(2)
TRSLTX	RTS/DTR Low Hold after Reset Inactive	0	Tcy − 20	(2)

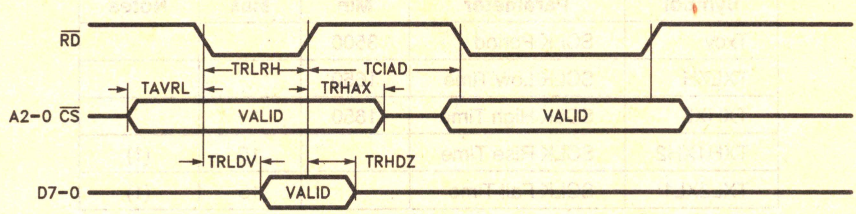


NOTES:
1. In case of CLK/X1 configured as X1, 1 Ms is required to guarantee crystal oscillator wake-up.
2. RTS/DTR are internally driven HIGH during RESET active time. The pin should be either left OPEN or externally driven LOW during RESET according to the required configuration of the system clock. These parameters specify the timing requirements on these pins, in case they are externally driven LOW during RESET.
The maximum spec on TRSLTX requires that the RTS/DTR pins not be forced later than TRSLTX maximum.

READ CYCLE SPECIFICATIONS

Symbol	Parameter	Min	Max	Notes
TRLRH	$\overline{RD}$ Active Width	2 Tcy + 65		
TAVRL	Address/$\overline{CS}$ Setup Time to $\overline{RD}$ Active	7		
TRHAX	Address/$\overline{CS}$ Hold Time after $\overline{RD}$ Inactive	0		
TRLDV	Data Out Valid Delay after $\overline{RD}$ Active	2Tcy + 65		
TCIAD	Command Inactive to Active Delay	Tcy + 15		(1)
TRHDZ	Data Out Float Delay after $\overline{RD}$ Inactive		40	

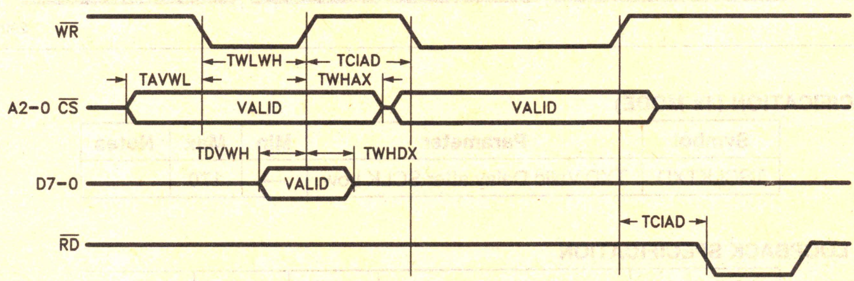
NOTE:
1. Command refers to either Read or Write signals.



290116–44

WRITE CYCLE SPECIFICATION

Symbol	Parameter	Min	Max	Notes
TWLWH	$\overline{WR}$ Active Width	2Tcy + 15		
TAVWL	Address $\overline{CS}$ Setup Time to $\overline{WR}$ Active	7		
TWHAX	Address and $\overline{CS}$ Hold Time after $\overline{WR}$	0		
TDVWH	Data in Setup Time to $\overline{WR}$ Inactive	90		
TWHDX	Data in Hold Time after $\overline{WR}$ Inactive	12		



290116–45

NOTE:
Many of the serial interface pins have more than one function; sometimes the different functions have different timings. In such a case, the timing of each function of a pin is given separately.

SCLK PIN SPECIFICATION—16x CLOCKING MODE

Symbol	Parameter	Min	Max	Notes
Txcy	SCLK Period	216		
TXLXH	SCLK Low Time	93		
TXHXL	SCLK High Time	93		
TXH1XH2	SCLK Rise Time		15	(1)
TXL2XL1	SCLK Fall Time		15	(1)

NOTE:
1. Rise/fall times are measured between 0.8V and 2.0V.

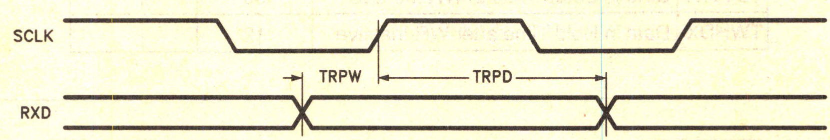
SCLK PIN SPECIFICATION—1x CLOCK MODE

Symbol	Parameter	Min	Max	Notes
Txcy	SCLK Period	3500		
TXLXH	SCLK Low Time	1650		
TXHXL	SCLK High Time	1650		
TXH1XH2	SCLK Rise Time		15	(1)
TXL2XL1	SCLK Fall Time		15	(1)

NOTE:
1. Rise/fall times are measured between 0.8V and 2.0V.

RXD SPECIFICATION (1x MODE)

Symbol	Parameter	Min	Max	Notes
TRPW	RXD Setup Time to SCLK High	250		
TRPD	RXD Hold Time After SCLK High	250		



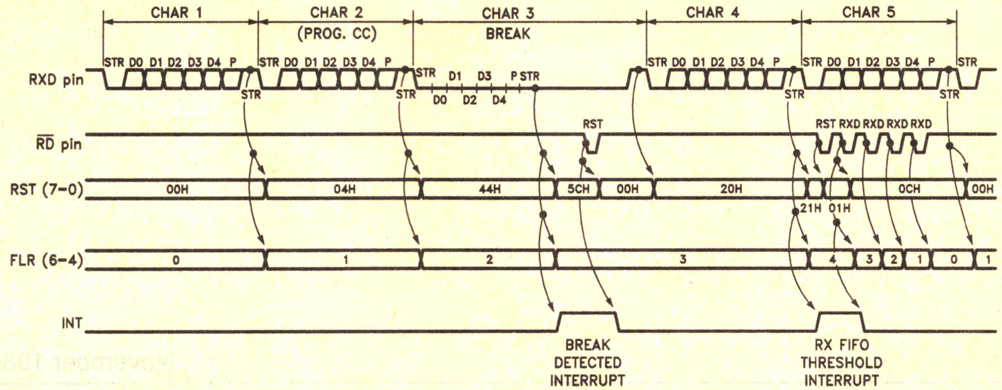
290116-46

TXD SPECIFICATION (1x MODE)

Symbol	Parameter	Min	Max	Notes
TSCLKTXD	TXD Valid Delay after SCLK Low	—	170	

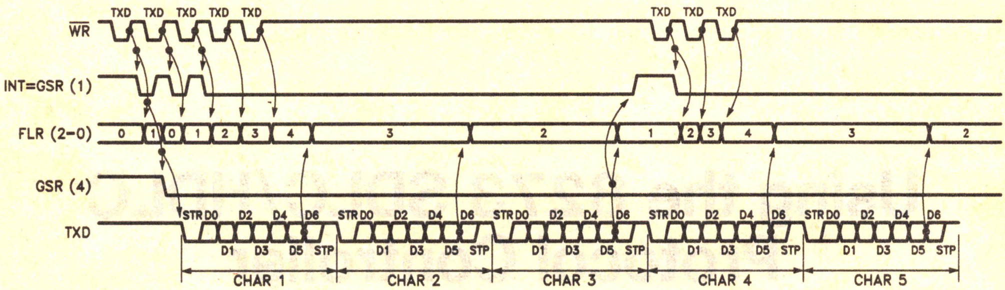
REMOTE LOOPBACK SPECIFICATION

Symbol	Parameter	Min	Max	Notes
TRXDTXD	TXD Delay after RXD	—	170	



290116-55

Receive Logic Diagram



290116-56

Transmit Logic Diagram

intel®

APPLICATION NOTE

AP-36

November 1986

Using the 8273 SDLC/HDLC Protocol Controller

JOHN BEASTON
MICROCOMPUTER APPLICATIONS

Order Number: 611001-001

INTRODUCTION

The Intel 8273 is a Data Communications Protocol Controller designed for use in systems utilizing either SDLC or HDLC (Synchronous or High-Level Data Link Control) protocols. In addition to the usual features such as full duplex operation, automatic Frame Check Sequence generation and checking, automatic zero bit insertion and deletion, and TTL compatibility found on other single component SDLC controllers, the 8273 features a frame level command structure, a digital phase locked loop, SDLC loop operation, and diagnostics.

The frame level command structure is made possible by the 8273's unique internal dual processor architecture. A high-speed bit processor handles the serial data manipulations and character recognition. A byte processor implements the frame level commands. These dual processors allow the 8273 to control the necessary byte-by-byte operation of the data channel with a minimum of CPU (Central Processing Unit) intervention. For the user this means the CPU has time to take on additional tasks. The digital phase locked loop (DPLL) provides a means of clock recovery from the received data stream on-chip. This feature, along with the frame level commands, makes SDLC loop operation extremely simple and flexible. Diagnostics in the form of both data and clock loopback are available to simplify board debug and link testing. The 8273 is a dedicated function peripheral in the MCS-80/85 Microcomputer family and as such, it interfaces to the 8080/8085 system with a minimum of external hardware.

This application note explains the 8273 as a component and shows its use in a generalized loop configuration and a typical 8085 system. The 8085 system was used to verify the SDLC operation of the 8273 on an actual IBM SDLC data communications link.

The first section of this application note presents an overview of the SDLC/HDLC protocols. It is fairly tutorial in nature and may be skipped by the more knowledgeable reader. The second section describes the 8273 from a functional standpoint with explanation of the block diagram. The software aspects of the 8273, including command examples, are discussed in the third section. The fourth and fifth sections discuss a loop SDLC configuration and the 8085 system respectively.

SDLC/HDLC OVERVIEW

SDLC is a protocol for managing the flow of information on a data communications link. In other words, SDLC can be thought of as an envelope—addressed, stamped, and containing an s.a.s.e.—in which information is transferred from location to location on a data communications link. (Please note that while SDLC is discussed specifically, all comments also apply to HDLC except where noted.) The link may be either point-to-point or multi-point, with the point-to-point configuration being either switched or nonswitched. The information flow may use either full or half duplex exchanges. With this many configurations supported, it is difficult to find a synchronous data communications application where SDLC would not be appropriate.

Aside from supporting a large number of configurations, SDLC offers the potential of a 2× increase in throughput over the presently most prevalent protocol: Bi-Sync. This performance increase is primarily due to two characteristics of SDLC: full duplex operation and the implied acknowledgement of transferred information. The performance increase due to full duplex operation is fairly obvious since, in SDLC, both stations can communicate simultaneously. Bi-Sync supports only half-duplex (two-way alternate) communication. The increase from implied acknowledgement arises from the fact that a station using SDLC may acknowledge previously received information while transmitting different information. Up to 7 messages may be outstanding before an acknowledgement is required. These messages may be acknowledged as a block rather than singly. In Bi-Sync, acknowledgements are unique messages that may not be included with messages containing information and each information message requires a separate acknowledgement. Thus the line efficiency of SDLC is superior to Bi-Sync. On a higher level, the potential of a 2× increase in performance means lower cost per unit of information transferred. Notice that the increase is not due to higher data link speeds (SDLC is actually speed independent), but simply through better line utilization.

Getting down to the more salient characteristics of SDLC; the basic unit of information on an SDLC link is that of the frame. The frame format is shown in Figure 1. Five fields comprise each frame: flag, address, control, information, and frame check sequence. The flag fields (F) form the boundary of the frame and all

Opening Flag	Address Field (A)	Control Field (C)	Information Field (I)	Frame Check Sequence (FCS)	Closing Flag
01111110	8 Bits	8 Bits	Any Length 0 to N Bits	16 Bits	01111110

Figure 1. SDLC Frame Format

other fields are positionally related to one of the two flags. All frames start with an opening flag and end with a closing flag. Flags are used for frame synchronization. They also may serve as time-fill characters between frames. (There are no intraframe time-fill characters in SDLC as there are in Bi-Sync.) The opening flag serves as a reference point for the address (A) and control (C) fields. The frame check sequence (FCS) is referenced from the closing flag. All flags have the binary configuration 01111110 (7EH).

SDLC is a bit-oriented protocol, that is, the receiving station must be able to recognize a flag (or any other special character) at any time, not just on an 8-bit boundary. This, of course, implies that a frame may be N-bits in length. (The vast majority of applications tend to use frames which are multiples of 8 bits long, however.)

The fact that the flag has a unique binary pattern would seem to limit the contents of the frame since a flag pattern might inadvertently occur within the frame. This would cause the receiver to think the closing flag was received, invalidating the frame. SDLC handles this situation through a technique called zero bit insertion. This techniques specifies that within a frame a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1s. Thus, no pattern of 01111110 is ever transmitted by chance. On the receiving end, after the opening flag is detected, the receiver removes any 0 following 5 consecutive 1s. The inserted and deleted 0s are not counted for error determination.

Before discussing the address field, an explanation of the roles of an SDLC station is in order. SDLC specifies two types of stations: primary and secondary. The primary is the control station for the data link and thus has responsibility of the overall network. There is only one predetermined primary station, all other stations on the link assume the secondary station role. In general, a secondary station speaks only when spoken to. In other words, the primary polls the secondaries for responses. In order to specify a specific secondary, each secondary is assigned a unique 8-bit address. It is this address that is used in the frame's address field.

When the primary transmits a frame to a specific secondary, the address field contains the secondary's address. When responding, the secondary uses its own address in the address field. The primary is never identified. This ensures that the primary knows which of many secondaries is responding since the primary may have many messages outstanding at various secondary stations. In addition to the specific secondary address, an address common to all secondaries may be used for various purposes. (An all 1s address field is usually used for this "All Parties" address.) Even though the primary may use this common address, the secondaries are expected to respond with their unique address. The address field is always the first 8 bits following the opening flag.

The 8 bits following the address field form the control field. The control field embodies the link-level control of SDLC. A detailed explanation of the commands and responses contained in this field is beyond the scope of this application note. Suffice it to say that it is in the control field that the implied acknowledgement is carried out through the use of frame sequence numbers. None of the currently available SDLC single chip controllers utilize the control field. They simply pass it to the processor for analysis. Readers wishing a more detailed explanation of the control field, or of SDLC in general, should consult the IBM documents referenced on the front page overleaf.

In some types of frames, an information field follows the control field. Frames used strictly for link management may or may not contain one. When an information field is used, it is unrestricted in both content and length. This code transparency is made possible because of the zero bit insertion mentioned earlier and the bit-oriented nature of SDLC. Even main memory core dumps may be transmitted because of this capability. This feature is unique to bit-oriented protocols. Like the control field, the information field is not interpreted by the SDLC device; it is merely transferred to and from memory to be operated on and interpreted by the processor.

The final field is the frame check sequence (FCS). The FCS is the 16 bits immediately preceding the closing flag. This 16-bit field is used for error detection through a Cyclic Redundancy Checkword (CRC). The 16-bit transmitted CRC is the complement of the remainder obtained when the A, C, and I fields are "divided" by a generating polynomial. The receiver accumulates the A, C, and I fields and also the FCS into its internal CRC register. At the closing flag, this register contains one particular number for an error-free reception. If this number is not obtained, the frame was received in error and should be discarded. Discarding the frame causes the station to not update its frame sequence numbering. This results in a retransmission after the station sends an acknowledgement from previous frames. [Unlike all other fields, the FCS is transmitted MSB (Most Significant Bit) first. The A, C, and I fields are transmitted LSB (Least Significant Bit) first.] The details of how the FCS is generated and checked is beyond the scope of this application note and since all single component SDLC controllers handle this function automatically, it is usually sufficient to know only that an error has or has not occurred. The IBM documents contain more detailed information for those readers desiring it.

The closing flag terminates the frame. When the closing flag is received, the receiver knows that the preceding 16 bits constitute the FCS and that any bits between the control field and the FCS constitute the information field.

SDLC does not support an interframe time-fill character such as the SYN character in Bi-Sync. If an unusual condition occurs while transmitting, such as data is not available in time from memory or CTS (Clear-to-Send) is lost from the modem, the transmitter aborts the frame by sending an Abort character to notify the receiver to invalidate the frame. The Abort character consists of eight contiguous 1s sent without zero bit insertion. Intraframe time-fill consists of either flags, Abort characters, or any combination of the two.

While the Abort character protects the receiver from transmitted errors, errors introduced by the transmission medium are discovered at the receiver through the FCS check and a check for invalid frames. Invalid frames are those which are not bounded by flags or are too short, that is, less than 32 bits between flags. All invalid frames are ignored by the receiver.

Although SDLC is a synchronous protocol, it provides an optional feature that allows its use on basically asynchronous data links—NRZI (Non-Return-to-Zero-Inverted) coding. NRZI coding specifies that the signal condition does not change for transmitting a binary 1, while a binary 0 causes a change of state. Figure 2 illustrates NRZI coding compared to the normal NRZ. NRZI coding guarantees that an active line will have a transition at least every 5-bit times; long strings of zeroes cause a transition every bit time, while long strings of 1s are broken up by zero bit insertion. Since asynchronous operation requires that the receiver sampling clock be derived from the received data, NRZI encoding plus zero bit insertion make the design of clock recovery circuitry easier.

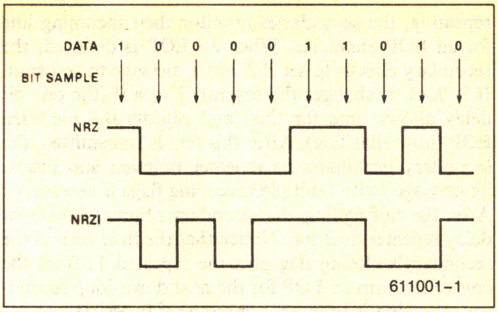


Figure 2. NRZI vs NRZ Encoding

All of the previous discussion has applied to SDLC on either point-to-point or multi-point data networks. SDLC (but not HDLC) also includes specification for a loop configuration. Figure 3 compares these three configurations. IBM uses this loop configuration in its 3650 Retail Store System. It consists of a single loop controller station with one or more down-loop secondary stations. Communications on a loop rely on the secondary stations repeating a received message down loop with a delay of one bit time. The reason for the one bit delay will be evident shortly.

Loop operation defines a new special character: the EOP (End-of-Poll) character which consists of a 0 followed by 7 contiguous, non-zero bit inserted, ones. After the loop controller transmits a message, it idles the line (sends all 1s). The final zero of the closing flag plus the first 7 1s of the idle form an EOP character. While

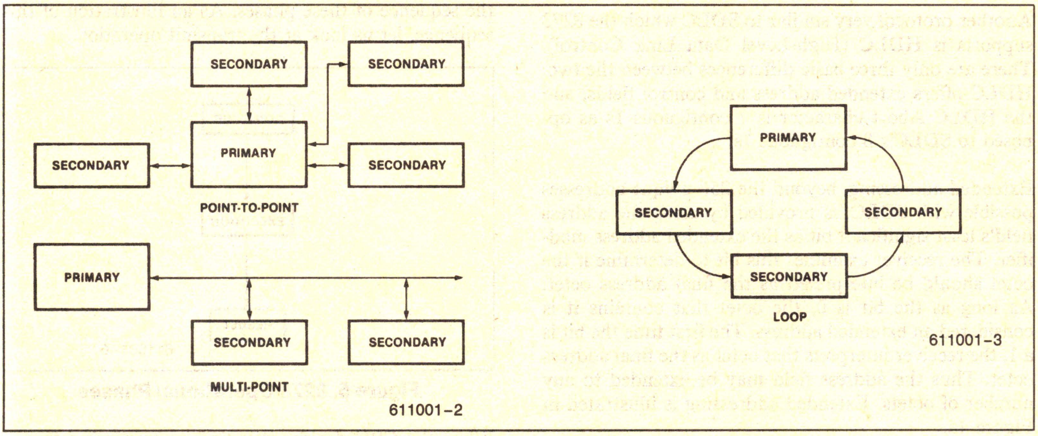


Figure 3. Network Configurations

repeating, the secondaries monitor their incoming line for an EOP character. When an EOP is detected, the secondary checks to see if it has a message to transmit. If it does, it changes the seventh 1 to a 0 (the one bit delay allows time for this) and repeats the modified EOP (now alias flag). After this flag is transmitted, the secondary terminates its repeater function and inserts its message (with multiple preceding flags if necessary). After the closing flag, the secondary resumes its one bit delay repeater function. Notice that the final zero of the secondary's closing flag plus the repeated 1s from the controller form an EOP for the next down-loop secondary, allowing it to insert a message if it desires.

One might wonder if the secondary missed any messages from the controller while it was inserting its own message. It does not. Loop operation is basically half-duplex. The controller waits until it receives an EOP before it transmits its next message. The controller's reception of the EOP signifies that the original message has propagated around the loop followed by any messages inserted by the secondaries. Notice that secondaries cannot communicate with one another directly, all secondary-to-secondary communication takes place by way of the controller.

Loop protocol does not utilize the normal Abort character. Instead, an abort is accomplished by simply transmitting a flag character. Down loop, the receiver sees the abort as a frame which is either too short (if the abort occurred early in the frame) or one with an FCS error. Either results in a discarded frame. For more details on loop operation, please refer to the IBM documents referenced earlier.

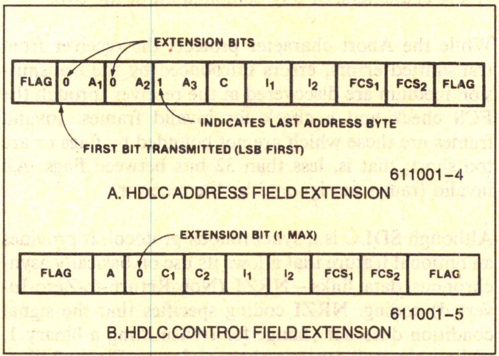
Another protocol very similar to SDLC which the 8273 supports is HDLC (High-Level Data Link Control). There are only three basic differences between the two: HDLC offers extended address and control fields, and the HDLC Abort character is 7 contiguous 1s as opposed to SDLC's 8 contiguous 1s.

Extended addressing, beyond the 256 unique addresses possible with SDLC, is provided by using the address field's least significant bit as the extended address modifier. The receiver examines this bit to determine if the octet should be interpreted as the final address octet. As long as the bit is 0, the octet that contains it is considered an extended address. The first time the bit is a 1, the receiver interprets that octet as the final address octet. Thus the address field may be extended to any number of octets. Extended addressing is illustrated in Figure 4a.

A similar technique is used to extend the control field although the extension is limited to only one extra control octet. Figure 4b illustrates control field extension.

Those readers not yet asleep may have noticed the similarity between the SDLC loop EOP character (a 0 followed by 7 1s) and the HDLC Abort (7 1s). This possible incompatibility is neatly handled by the HDLC protocol not specifying a loop configuration.

This completes our brief discussion of the SDLC/HDLC protocols. Now let us turn to the 8273 in particular and discuss its hardware aspects through an explanation of the block diagram and generalized system schematics.



A. HDLC ADDRESS FIELD EXTENSION

B. HDLC CONTROL FIELD EXTENSION

Figure 4

BASIC 8273 OPERATION

It will be helpful for the following discussions to have some idea of the basic operation of the 8273. Each operation, whether it is a frame transmission, reception or port read, etc., is comprised of three phases: the Command, Execution, and Result phases. Figure 5 shows the sequence of these phases. As an illustration of this sequence, let us look at the transmit operation.

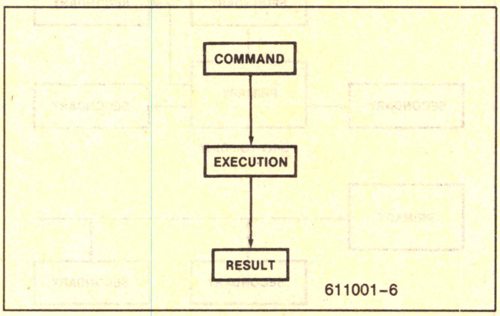


Figure 5. 8273 Operational Phases

When the CPU decides it is time to transmit a frame, the Command phase is entered by the CPU issuing a Transmit Frame command to the 8273. It is not sufficient to just instruct the 8273 to transmit. The frame level command structure sometimes requires more information such as frame length and address and control field content. Once this additional information is sup-

plied, the Command phase is complete and the Execution phase is entered. It is during the Execution phase that the actual operation, in this case a frame transmission, takes place. The 8273 transmits the opening flag, A and C fields, the specified number of I field bytes, inserts the FCS, and closes with the closing flag. Once the closing flag is transmitted, the 8273 leaves the Execution phase and begins the Result phase. During the Result phase the 8273 notifies the CPU of the outcome of the command by supplying interrupt results. In this case, the results would be either that the frame is complete or that some error condition causes the transmission to be aborted. Once the CPU reads all of the results (there is only one for the Transmit Frame command), the Result phase and consequently the operation, is complete. Now that we have a general feeling for the operation of the 8273, let us discuss the 8273 in detail.

HARDWARE ASPECTS OF THE 8273

The 8273 block diagram is shown in Figure 6. It consists of two major interfaces: the CPU module interface and the modem interface. Let's discuss each interface separately.

CPU Interface

The CPU interface consists of four major blocks: Control/Read/Write logic (C/R/W), internal registers, data transfer logic, and data bus buffers.

The CPU module utilizes the C/R/W logic to issue commands to the 8273. Once the 8273 receives a command and executes it, it returns the results (good/bad completion) of the command by way of the C/R/W logic. The C/R/W logic is supported by seven registers which are addressed via the A_0, A_1, $\overline{RD}$, and $\overline{WR}$ signals, in addition to $\overline{CS}$. The A_0 and A_1 signals are generally derived from the two low order bits of the CPU module address bus while $\overline{RD}$ and $\overline{WR}$ are the normal I/O Read and Write signals found on the system control bus. Figure 7 shows the address of each register using the C/R/W logic. The function of each register is defined as follows:

Address Inputs		Control Inputs	
A_1	A_0	$\overline{CS}\bullet\overline{RD}$	$\overline{CS}\bullet\overline{WR}$
0	0	Status	Command
0	1	Result	Parameter
1	0	TxI/R	Test Mode
1	1	RxI/R	—

Figure 7. 8273 Register Selection

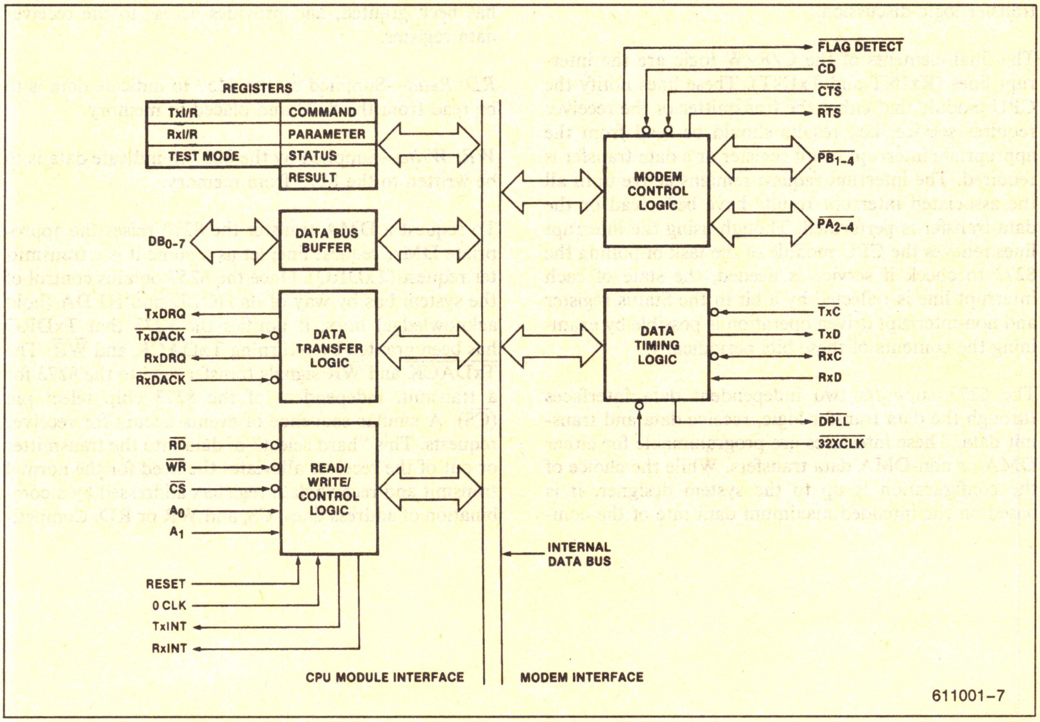


Figure 6. 8273 Block Diagram

Command—8273 operations are initiated by writing the appropriate command byte into this register.

Parameter—Many commands require more information than found in the command itself. This additional information is provided by way of the parameter register.

Immediate Result (Result)—The completion information (results) for commands which execute immediately are provided in this register.

Transmit Interrupt Result (TxI/R)—Results of transmit operations are passed to the CPU in this register.

Receiver Interrupt Result (RxI/R)—Receive operation results are passed to the CPU via this register.

Status—The general status of the 8273 is provided in this register. The Status register supplies the handshaking necessary during various phases of the 8273 operation.

Test Mode—This register provides a software reset function for the 8273.

The commands, parameters, and bit definition of these registers are discussed in the following software section. Notice that there are not specific transmit or receive data registers. This feature is explained in the data transfer logic discussion.

The final elements of the C/R/W logic are the interrupt lines (RxINT and TxINT). These lines notify the CPU module that either the transmitter or the receiver requires service; i.e., results should be read from the appropriate interrupt result register or a data transfer is required. The interrupt request remains active until all the associated interrupt results have been read or the data transfer is performed. Though using the interrupt lines relieves the CPU module of the task of polling the 8273 to check if service is needed, the state of each interrupt line is reflected by a bit in the Status register and non-interrupt driven operation is possible by examining the contents of these bits periodically.

The 8273 supports two independent data interfaces through the data transfer logic; receive data and transmit data. These interfaces are programmable for either DMA or non-DMA data transfers. While the choice of the configuration is up to the system designer, it is based on the intended maximum data rate of the communications channel. Figure 8 illustrates the transfer rate of data bytes that are acquired by the 8273 based on link data rate. Full-duplex data rates above 9600 baud usually require DMA. Slower speeds may or may not require DMA depending on the task load and interrupt response time of the processor.

Figure 9 shows the 8273 in a typical DMA environment. Notice that a separate DMA controller, in this case the Intel 8257, is required. The DMA controller supplies the timing and addresses for the data transfers while the 8273 manages the requesting of transfers and the actual counting of the data block lengths. In this case, elements of the data transfer interface are:

TxDRQ: Transmit DMA Request—Asserted by the 8273, this line requests a DMA transfer from memory to the 8273 for transmit.

TxDACK: Transmit DMA Acknowledge—Returned by the 8257 in response to TxDRQ, this line notifies the 8273 that a request has been granted, and provides access to the transmitter data register.

RxDRQ: Receive DMA Request—Asserted by the 8273, it requests a DMA transfer from the 8273 to memory for a receive operation.

RxDACK: Receive DMA Acknowledge—Returned by the 8257, it notifies the 8273 that a receive DMA cycle has been granted, and provides access to the receiver data register.

RD: Read—Supplied by the 8257 to indicate data is to be read from the 8273 and placed in memory.

WR: Write—Supplied by the 8257 to indicate data is to be written to the 8273 from memory.

To request a DMA transfer the 8273 raises the appropriate DMA request line; let us assume it is a transmitter request (TxDRQ). Once the 8257 obtains control of the system bus by way of its HOLD and HLDA (hold acknowledge) lines, it notifies the 8273 that TxDRQ has been granted by returning $\overline{\text{TxDACK}}$ and $\overline{\text{WR}}$. The $\overline{\text{TxDACK}}$ and $\overline{\text{WR}}$ signals transfer data to the 8273 for a transmit, independent of the 8273 chip select pin ($\overline{\text{CS}}$). A similar sequence of events occurs for receiver requests. This "hard select" of data into the transmitter or out of the receiver alleviates the need for the normal transmit and receive data registers addressed by a combination of address lines, CS, and WR or RD. Competi-

tive devices that do not have this "hard select" feature require the use of an external multiplexer to supply the correct inputs for register selection during DMA. (Do not forget that the SDLC controller sees both the addresses and control signals supplied by the DMA controller during DMA cycles.) Let us look at typical frame transmit and frame receive sequences to better see how the 8273 truly manages the DMA data transfer.

Before a frame can be transmitted, the DMA controller is supplied, by the CPU, the starting address for the desired information field. The 8273 is then commanded to transmit a frame. (Just how this is done is covered later during our software discussion.) After the command, but before transmission begins, the 8273 needs a little more information (parameters). Four parameters are required for the transmit frame command: the address field byte, the control field byte, and two bytes which are the least significant and most significant bytes of the information field byte length. Once all four parameters are loaded, the 8273 makes RTS (Request-to-Send) active and waits for CTS (Clear-to-Send) to go active. Once CTS is active, the 8273 starts the frame transmission. While the 8273 is transmitting the opening flag, address field, and control field; it starts making transmitter DMA requests. These requests continue at character (byte) boundaries until the pre-loaded number of bytes of information field have been transmitted. At this point the requests stop, the FCS and closing flag are transmitted, and the TxINT line is raised, signaling the CPU that the frame transmission is complete. Notice that after the initial command and parameter loading, absolutely no CPU intervention was required (since DMA is used for data transfers) until the entire frame was transmitted. Now let's look at a frame reception.

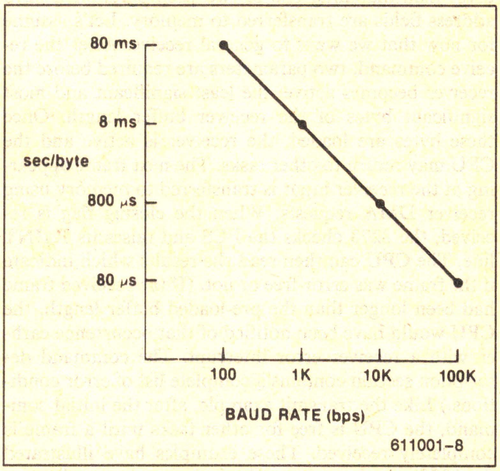


Figure 8. Byte Transfer Rate vs Baud Rate

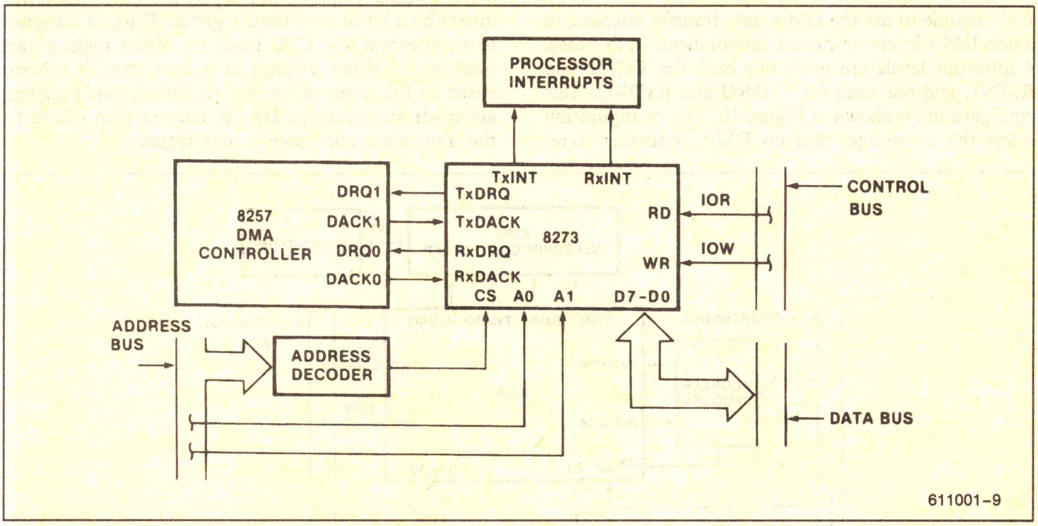


Figure 9. DMA, Interrupt-Driven System

The receiver operation is very similar. Like the initial transmit sequence, the DMA controller is loaded with a starting address for a receiver data buffer and the 8273 is commanded to receive. Unlike the transmitter, there are two different receive commands: General Receive, where all received frames are transferred to memory, and Selective Receive, where only frames having an address field matching one of two preprogrammed 8273 address fields are transferred to memory. Let's assume for now that we want to general receive. After the receive command, two parameters are required before the receiver becomes active: the least significant and most significant bytes of the receiver buffer length. Once these bytes are loaded, the receiver is active and the CPU may return to other tasks. The next frame appearing at the receiver input is transferred to memory using receiver DMA requests. When the closing flag is received, the 8273 checks the FCS and raises its RxINT line. The CPU can then read the results which indicate if the frame was error-free or not. (If the received frame had been longer than the pre-loaded buffer length, the CPU would have been notified of that occurrence earlier with a receiver error interrupt. The command description section contains a complete list of error conditions.) Like the transmit example, after the initial command, the CPU is free for other tasks until a frame is completely received. These examples have illustrated the 8273's management of both the receiver and transmitter DMA channels.

It is possible to use the DMA data transfer interface in a non-DMA interrupt-driven environment. In this case, 4 interrupt levels are used: one each for TxINT and RxINT, and one each for TxDRQ and RxDRQ. This configuration is shown in Figure 10. This configuration offers the advantages that no DMA controller is required and data requests are still separated from result (completion) requests. The disadvantages of the configuration are that 4 interrupt levels are required and that the CPU must actually supply the data transfers. This, of course, reduces the maximum data rate compared to the configuration based strictly on DMA. This system could use an Intel 8259 8-level Priority Interrupt Controller to supply a vectored CALL (subroutine) address based on requests on its inputs. The 8273 transmitter and receiver make data requests by raising the respective DRQ line. The CPU is interrupted by the 8259 and vectored to a data transfer routine. This routine either writes (for transmit) or reads (for receive) the 8273 using the respective TxDACK or RxDACK line. The DACK lines serve as "hard" chip selects into and out of the 8273. $\overline{\text{TxDACK}}$ + $\overline{\text{WR}}$ writes data into the 8273 for transmit. $\overline{\text{RxDACK}}$ + $\overline{\text{RD}}$ reads data from the 8273 for receive.) The CPU is notified of operation completion and results by way of TxINT and RxINT lines. Using the 8273, and the 8259, in this way, provides a very effective, yet simple, interrupt-driven interface.

Figure 11 illustrates a system very similar to that described above. This system utilizes the 8273 in a non-DMA data transfer mode as opposed to the two DMA approaches shown in Figures 9 and 10. In the non-DMA case, data transfer requests are made on the TxINT and RxINT lines. The DRQ lines are not used. Data transfer requests are separated from result requests by a bit in the Status register. Thus, in response to an interrupt, the CPU reads the Status register and branches to either a result or a data transfer routine based on the status of one bit. As before, data transfers are made via using the DACK lines as chip selects to the transmitter and receiver data registers.

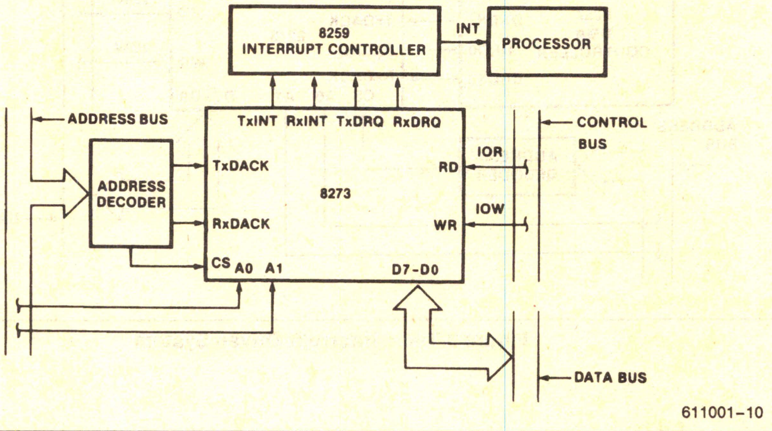


Figure 10. Interrupt-Based DMA System

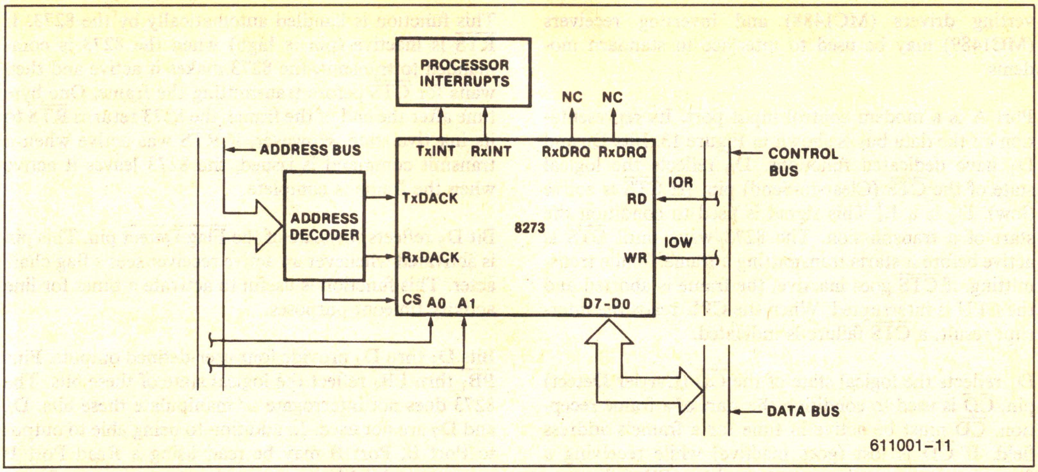


Figure 11. Non-DMA Interrupt-Driven System

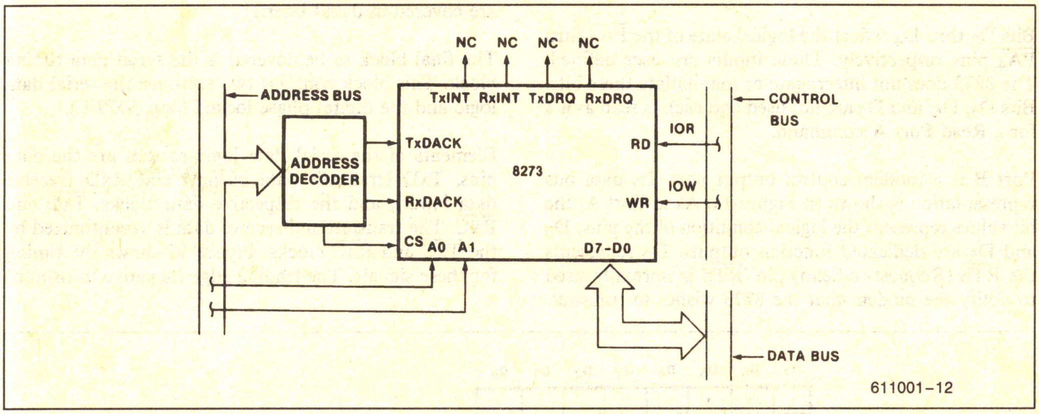


Figure 12. Polled System

Figure 12 illustrates the simplest system of all. This system utilizes polling for all data transfers and results. Since the interrupt pins are reflected in bits in the Status register, the software can read the Status register periodically looking for one of these to be set. If it finds an INT bit set, the appropriate Result Available bit is examined to determine if the "interrupt" is a data transfer or completion result. If a data transfer is called for, the DACK line is used to enter or read the data from the 8273. If the interrupt is a completion result, the appropriate result register is read to determine the good/bad completion of the operation.

The actual selection of either DMA or non-DMA modes is controlled by a command issued during initialization. This command is covered in detail during the software discussion.

The final block of the CPU module interface is the Data Bus Buffer. This block supplies the tri-state, bidirectional data bus interface to allow communication to and from the 8273.

Modem Interface

As the name implies, the modem interface is the modem side of the 8273. It consists of two major blocks: the modem control block and the serial data timing block.

The modem control block provides both dedicated and user-defined modem control functions. All signals supported by this interface are active low so that EIA in-

verting drivers (MC1488) and inverting receivers (MC1489) may be used to interface to standard modems.

Port A is a modem control input port. Its representation on the data bus is shown in Figure 13. Bits D_0 and D_1 have dedicated functions. D_0 reflects the logical state of the $\overline{CTS}$ (Clear-to-Send) pin. [If $\overline{CTS}$ is active (low), D_0 is a 1.] This signal is used to condition the start of a transmission. The 8273 waits until $\overline{CTS}$ is active before it starts transmitting a frame. While transmitting, if $\overline{CTS}$ goes inactive, the frame is aborted and the CPU is interrupted. When the CPU reads the interrupt result, a $\overline{CTS}$ failure is indicated.

D_1 reflects the logical state of the $\overline{CD}$ (Carrier Detect) pin. $\overline{CD}$ is used to condition the start of a frame reception. $\overline{CD}$ must be active in time for a frame's address field. If $\overline{CD}$ is lost (goes inactive) while receiving a frame, an interrupt is generated with a $\overline{CD}$ failure result. $\overline{CD}$ may go inactive between frames.

Bits D_2 thru D_4 reflect the logical state of the $\overline{PA_2}$ thru $\overline{PA_4}$ pins respectively. These inputs are user defined. The 8273 does not interrogate or manipulate these bits. Bits D_5, D_6, and D_7 are not used and each is read as a 1 for a Read Port A command.

Port B is a modem control output port. Its data bus representation is shown in Figure 14. As in Port A, the bit values represent the logical condition of the pins. D_0 and D_5 are dedicated function outputs. D_0 represents the $\overline{RTS}$ (Request-to-Send) pin. $\overline{RTS}$ is normally used to notify the modem that the 8273 wishes to transmit. This function is handled automatically by the 8273. If $\overline{RTS}$ is inactive (pin is high) when the 8273 is commanded to transmit, the 8273 makes it active and then waits for $\overline{CTS}$ before transmitting the frame. One byte time after the end of the frame, the 8273 returns $\overline{RTS}$ to its inactive state. However, if $\overline{RTS}$ was active when a transmit command is issued, the 8273 leaves it active when the frame is complete.

Bit D_5 reflects the state of the $\overline{\text{Flag Detect}}$ pin. This pin is activated whenever an active receiver sees a flag character. This function is useful to activate a timer for line activity timeout purposes.

Bits D_1 thru D_4 provide four user-defined outputs. Pins $\overline{PB_1}$ thru $\overline{PB_4}$ reflect the logical state of these bits. The 8273 does not interrogate or manipulate these bits. D_6 and D_7 are not used. In addition to being able to output to Port B, Port B may be read using a Read Port B command. All Modem control output pins are forced high on reset. (All commands mentioned in this section are covered in detail later.)

The final block to be covered is the serial data timing block. This block contains two sections: the serial data logic and the digital phase locked loop (DPLL).

Elements of the serial data logic section are the data pins, TxD (transmit data output) and RxD (receive data input), and the respective data clocks, $\overline{TxC}$ and $\overline{RxC}$. The transmit and receive data is synchronized by the $\overline{TxC}$ and $\overline{RxC}$ clocks. Figure 15 shows the timing for these signals. The leading edge (negative transition)

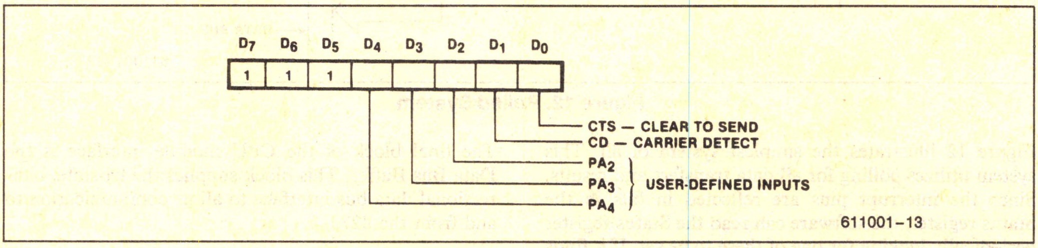


Figure 13. Port A (Input) Bit Definition

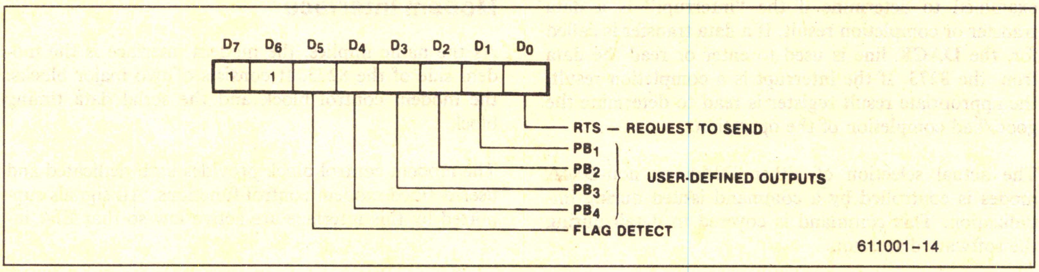


Figure 14. Port B (Output) Bit Definition

of $\overline{\text{TxC}}$ generates new transmit data and the trailing edge (positive transition) of $\overline{\text{RxC}}$ is used to capture the receive data.

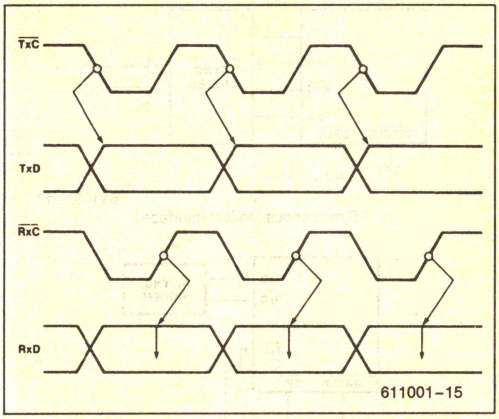


Figure 15. Transmit/Receive Timing

It is possible to reconfigure this section under program control to perform diagnostic functions; both data and clock loopback are available. In data loopback mode, the TxD pin is internally routed to the RxD pin. This allows simple board checkout since the CPU can send an SDLC message to itself. (Note that transmitted data will still appear on the TxD pin.)

When data loopback is utilized, the receiver may be presented incorrect sample timing ($\overline{\text{RxC}}$) by the external circuitry. Clock loopback overcomes this problem by allowing the internal routing of $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$. Thus the same clock used to transmit the data is used to receive it. Examination of Figure 15 shows that this method ensures bit synchronism. The final element of the serial data logic is the Digital Phase Locked Loop.

The DPLL provides a means of clock recovery from the received data stream. This feature allows the 8273 to interface without external synchronizing logic to low cost asynchronous modems (modems which do not supply clocks). It also makes the problem of clock timing in loop configurations trivial.

To use the DPLL, a clock at 32 times the required baud rate must be supplied to the $\overline{32\times \text{ CLK}}$ pin. This clock provides the interval that the DPLL samples the received data. The DPLL uses the 32× clock and the received data to generate a pulse at the $\overline{\text{DPLL}}$ output pin. This $\overline{\text{DPLL}}$ pulse is positioned at the nominal center of the received data bit cell. Thus the $\overline{\text{DPLL}}$ output may be wired to $\overline{\text{RxC}}$ and/or $\overline{\text{TxC}}$ to supply the data timing. The exact position of the pulse is varied depending on the line noise and bit distortion of the received data. The adjustment of the $\overline{\text{DPLL}}$ position is determined according to the rules outlined in Figure 16.

Adjustments to the sample phase of $\overline{\text{DPLL}}$ with respect to the received data is made in discrete increments. Referring to Figure 16, following the occurrence of $\overline{\text{DPLL}}$

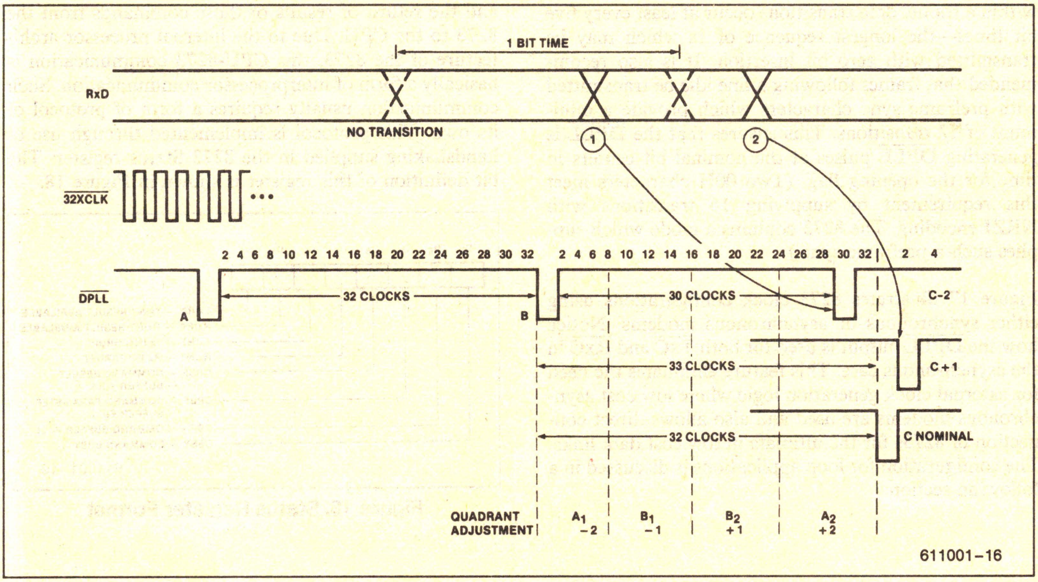


Figure 16. DPLL Phase Adjustments

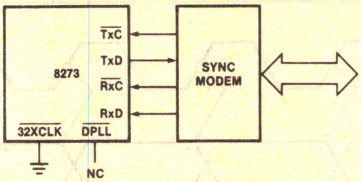
pulse A, the DPLL counts $\overline{32\times \text{ CLK}}$ pulses and examines the received data for a data edge. Should no edge be detected in 32 pulses, the $\overline{\text{DPLL}}$ positions the next $\overline{\text{DPLL}}$ pulse (B) at 32 clock pulses from pulse A. Since no new phase information is contained in the data stream, the sample phase is assumed to be at nominal 1× baud rate. Now assume a data edge occurs after $\overline{\text{DPLL}}$ pulse B. The distance from B to the next pulse C is influenced according to which quadrant (A_1, B_1, B_2, or A_2) the data edge falls in. (Each quadrant represents 8 $\overline{32\times \text{ CLK}}$ times.) For example, if the edge is detected in quadrant A_1, it is apparent that pulse B was too close to the data edge and the time to the next pulse must be shortened. The adjustment for quadrant A_1 is specified as −2. Thus, the next $\overline{\text{DPLL}}$ pulse, pulse C, is positioned 32 − 2 or 30 $\overline{32\times \text{ CLK}}$ pulses following $\overline{\text{DPLL}}$ pulse B. This adjustment moves pulse C closer to the nominal bit center of the next received data cell. A data edge occurring in quadrant B_2 would have caused the adjustment to be small, namely 32 + 1 or 33 $\overline{32\times \text{ CLK}}$ pulses. Using this technique, the $\overline{\text{DPLL}}$ pulse converges to the nominal bit center within 12 data transitions, worse case—4-bit times adjusting through quadrant A_1 or A_2 and 8-bit times adjusting through B_1 or B_2.

When the receive data stream goes idle after 15 ones, DPLL pulses are generated at 32 pulse intervals of the 32× CLK. This feature allows the DPLL pulses to be used as both transmitter and receiver clocks.

In order to guarantee sufficient transitions of the received data to enable the $\overline{\text{DPLL}}$ to lock, NRZI encoding of the data is recommended. This ensures that, within a frame, data transitions occur at least every five bit times—the longest sequence of 1s which may be transmitted with zero bit insertion. It is also recommended that frames following a line idle be transmitted with preframe sync characters which provide a minimum of 12 transitions. This ensures that the $\overline{\text{DPLL}}$ is generating $\overline{\text{DPLL}}$ pulses at the nominal bit centers in time for the opening flag. (Two 00H characters meet this requirement by supplying 16 transitions with NRZI encoding. The 8273 contains a mode which supplies such a preframe sync.)

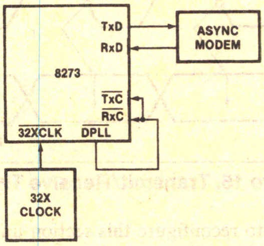
Figure 17 illustrates 8273 clock configurations using either synchronous or asynchronous modems. Notice how the $\overline{\text{DPLL}}$ output is used for both $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$ in the asynchronous case. This feature eliminates the need for external clock generation logic where low cost asynchronous modems are used and also allows direct connection of 8273s for the ultimate in low cost data links. The configuration for loop applications is discussed in a following section.

This completes our discussion of the hardware aspects of the 8273. Its software aspects are now discussed.



611001-17

Synchronous Modem Interface



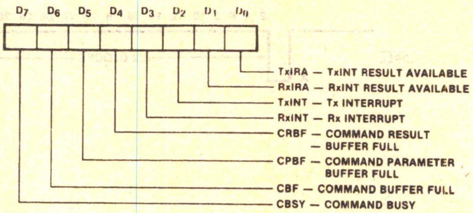
611001-18

Asynchronous Modem Interface

Figure 17. Serial Data Timing Configuration

SOFTWARE ASPECTS OF THE 8273

The software aspects of the 8273 involve the communication of both commands from the CPU to the 8273 and the return of results of those commands from the 8273 to the CPU. Due to the internal processor architecture of the 8273, this CPU-8273 communication is basically a form of interprocessor communication. Such communication usually requires a form of protocol of its own. This protocol is implemented through use of handshaking supplied in the 8273 Status register. The bit definition of this register is shown in Figure 18.



611001-19

Figure 18. Status Register Format

CBSY: Command Busy—CBSY indicates when the 8273 is in the command phase. CBSY is set when the CPU writes a command into the Command register, starting the Command phase. It is reset when the last parameter is deposited in the Parameter register and accepted by the 8273, completing the Command phase.

CBF: Command Buffer Full—When set, this bit indicates that a byte is present in the Command register. This bit is normally not used.

CPBF: Command Parameter Buffer Full—This bit indicates that the Parameter register contains a parameter. It is set when the CPU deposits a parameter in the Parameter register. It is reset when the 8273 accepts the parameter.

CRBF: Command Result Buffer Full—This bit is set when the 8273 places a result from an immediate type command in the Result register. It is reset when the CPU reads the result from the Result register.

RxINT: Receiver Interrupt—The state of the RxINT pin is reflected by this bit. RxINT is set by the 8273 whenever the receiver needs servicing. RxINT is reset when the CPU reads the results or performs the data transfer.

TxINT: Transmitter Interrupt—This bit is identical to RxINT except action is initiated based on transmitter interrupt sources.

RxIRA: Receiver Interrupt Result Available—RxIRA is set when the 8273 places an interrupt result byte into the RxI/R register. RxIRA is reset when the CPU reads the RxI/R register.

TxIRA: Transmitter Interrupt Result Available—TxIRA is the corresponding Result Available bit for the transmitter. It is set when the 8273 places an interrupt result byte in the TxI/R register and reset when the CPU reads the register.

The significance of each of these bits will be evident shortly. Since the software requirements of each 8273 phase are essentially independent, each phase is covered separately.

Command Phase Software

Recalling the Command phase description in an earlier section, the CPU starts the Command phase by writing a command byte into the 8273 Command register. If further information about the command is required by the 8273, the CPU writes this information into the Parameter register. Figure 19 is a flowchart of the Command phase. Notice that the CBSY and CPBF bits of the Status register are used to handshake the command and parameter bytes. Also note that the chart shows

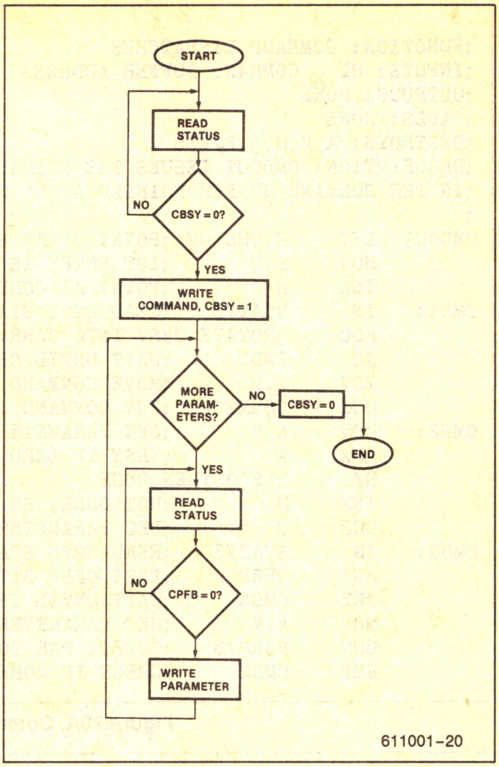


Figure 19. Command Phase Flowchart

that a command may not be issued if the Status register indicates the 8273 is busy (CBSY = 1). If a command is issued while CBSY = 1, the original command is overwritten and lost. (Remember that CBSY signifies the command phase is in progress and not the actual execution of the command.) The flowchart also includes a Parameter buffer full check. The CPU must wait until CPBF = 0 before writing a parameter to the Parameter register. If a parameter is issued while CPBF = 1, the previous parameter is overwritten and lost. An example of command output assembly language software is provided in Figure 20a. This software assumes that a command buffer exists in memory. The buffer is pointed at by the HL register. Figure 20b shows the command buffer structure.

The 8273 is a full duplex device, i.e., both the transmitter and receiver may be executing commands or passing interrupt results at any given time. (Separate Rx and Tx interrupt pins and result registers are provided for this reason.) However, there is only one Command register. Thus, the Command register must be used for only one command sequence at a time and the transmitter and receiver may never be simultaneously in a command phase. A detailed description of the commands and their parameters is presented in a following section.

```
;FUNCTION: COMMAND DISPATCHER
;INPUTS: HL - COMMAND BUFFER ADDRESS
;OUTPUTS: NONE
;CALLS: NONE
;DESTROYS: A,B,H,L,F/F'S
;DESCRIPTION: CMDOUT ISSUES THE COMMAND + PARAMETERS
;IN THE COMMAND BUFFER POINTED AT BY HL
;
CMDOUT: LXI     H,CMDBUF ;POINT HL AT BUFFER
        MOV     B,M      ;1ST ENTRY IS PAR. COUNT
        INX     H        ;POINT AT COMMAND BYTE
CMD1:   IN      STAT73   ;READ 8273 STATUS
        RLC      ;ROTATE CBSY INTO CARRY
        JC      CMD1     ;WAIT UNTIL CBSY=0
        MOV     A,M      ;MOVE COMMAND BYTE TO A
        OUT     COMM73   ;PUT COMMAND IN COMMAND REG
CMD2:   MOV     A,B      ;GET PARAMETER COUNT
        ANA     A        ;TEST IF ZERO
        RZ       ;IF 0 THEN DONE
        INX     H        ;NOT DONE, SO POINT AT NEXT PAR
        DCR     B        ;DEC PARAMETER COUNT
CMD3:   IN      STAT73   ;READ 8273 STATUS
        ANI     CPBF     ;TEST CPBF BIT
        JNZ     CMD3     ;WAIT UNTIL CPBF IS 0
        MOV     A,M      ;GET PARAMETER FROM BUFFER
        OUT     PARM73   ;OUTPUT PAR TO PARAMETER REG
        JMP     CMD2     ;CHECK IF MORE PARAMETERS
```

Figure 20A. Command Phase Software

+4	PARAMETER 3	
+3	PARAMETER 2	
+2	PARAMETER 1	
+1	COMMAND	
CMDBUF:	PARAMETER COUNT	← HL

Figure 20B. Command Buffer Format

Execution Phase Software

During the Execution phase, the operation specified by the Command phase is performed. If the system utilizes DMA for data transfers, there is no CPU involvement during this phase, so no software is required. If non-DMA data transfers are used, either interrupts or polling is used to signal a data transfer request.

For interrupt-driven transfers the 8273 raises the appropriate INT pin. When responding to the interrupt,

the CPU must determine whether it is a data transfer request or an interrupt signaling that an operation is complete and results are available. The CPU determines the cause by reading the Status register and interrogating the associated IRA (Interrupt Result Available) bit (TxIRA for TxINT and RxIRA for RxINT). If the IRA = 0, the interrupt is a data transfer request. If the IRA = 1, an operation is complete and the associated Interrupt Result register must be read to determine the completion status (good/bad/etc.). A software interrupt handler implementing the above sequence is presented as part of the Result phase software.

When polling is used to determine when data transfers are required, the polling routine reads the Status register looking for one of the INT bits to be set. When a set INT bit is found, the corresponding IRA bit is examined. Like in the interrupt-driven case, if the IRA = 0, a data transfer is required. If IRA = 1, an operation is complete and the Interrupt Result register needs to be read. Again, example polling software is presented in the next section.

Result Phase Software

During the Result phase the 8273 notifies the CPU of the outcome of a command. The Result phase is initiated by either a successful completion of an operation or an error detected during execution. Some commands such as reading or writing the I/O ports provide immediate results, that is, there is essentially no delay from the issuing of the command and when the result is available. Other commands such as frame transmit, take time to complete so their result is not available immediately. Separate result registers are provided to distinguish these two types of commands and to avoid interrupt handling for simple results.

Immediate results are provided in the Result register. Validity of information in this register is indicated to the CPU by way of the CRBF bit in the Status register. When the CPU completes the Command phase of an immediate command, it polls the Status register waiting until CRBF = 1. When this occurs, the CPU may read the Result register to obtain the immediate result. The Result register provides only the results from immediate commands.

Example software for handling immediate results is shown in Figure 21. The routine returns with the result in the accumulator. The CPU then uses the result as is appropriate.

All non-immediate commands deal with either the transmitter or receiver. Results from these commands are provided in the TxI/R (Transmit Interrupt Result) and RxI/R (Receive Interrupt Result) registers respectively. Results in these registers are conveyed to the CPU by the TxIRA and RxIRA bits of the status register. Results of non-immediate commands consist of one byte result interrupt code indicating the condition for the interrupt and, if required, one or more bytes supplying additional information. The interrupt codes and the meaning of the additional results are covered following the detailed command description.

Non-immediate results are passed to the CPU in response to either interrupts or polling of the Status register. Figure 22 illustrates an interrupt-driven result handler. (Please note that all of the software presented in this application note is not optimized for either speed or code efficiency. They are provided as a guide and to illustrate concepts.) This handler provides for interrupt-driven data transfers as was promised in the last section. Users employing DMA-based transfers do not

```
;FUNCTION: IMDRLT
;INPUTS: NONE
;OUTPUTS: RESULT REGISTER IN A
;CALLS: NONE
;DESTROYS: A, F/F'S
;DESCRIPTION: IMDRLT IS CALLED AFTER A CMDOUT FOR AN
;IMMEDIATE COMMAND TO READ THE RESULT REGISTER
;
IMDRLT: IN    STAT 73   ;READ 8273 STATUS
        ANI   CRBF      ;TEST IF RESULT REG READY
        JZ    IMDRLT    ;WAIT IF CRBF=0
        IN    RESL73    ;READ RESULT REGISTER
        RET   ;RETURN
```

Figure 21. Immediate Result Handler

```
;FUNCTION: RXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: RCRBUF, RCVPNT
;CALLS: NONE
;OUTPUTS: RCRBUF, RCVPNT
;DESTROYS: NOTHING
;DESCRIPTION: RXI IS ENTERED AT A RECEIVER INTERRUPT.
;THE INTERRUPT IS TESTED FOR DATA TRANSFER (IRA=0)
;OR RESULT (IRA=1).  FOR DATA TRANSFER, THE DATA IS
;PLACED IN A BUFFER AT RCVPNT. RESULTS ARE PLACED IN
;A BUFFER AT RCRBUF.
;A FLAG(RXFLAG) IS SET IF THE INTERRUPT WAS A RESULT.
;(DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*) AND
;MAYBE ELIMINATED BY USERS USING DMA.
;
RXI:    PUSH    H        ;SAVE HL
        PUSH    PSW      ;SAVE PSW
        PUSH    B        ;SAVE B
        IN      STAT73   ; (*) READ 8273 STATUS
        ANI     RXIRA    ; (*) TEST IRA BIT
        JZ      RXI2     ; (*) IF 0, DATA TRANSFER NEEDED
RXI1:   LHLD    RCRBUF   ;GET RESULT BUFFER POINTER
        IN      STAT73   ;READ 8273 STATUS AGAIN
        ANI     RXINT    ;TEST INT BIT
        JZ      RXI4     ;IF 0, THEN DONE
        IN      STAT73   ;READ 8273 STATUS AGAIN
        ANI     RXIRA    ;TEST IRA AGAIN
        JZ      RXI1     ;LOOP UNTIL RESULT IS READY
        IN      RXIR73   ;READY, READ RXI/R
        MOV     M,A      ;STORE RESULT IN BUFFER
        INX     H        ;BUMP RESULT POINTER
        SHLD    RCRBUF   ;RESTORE BUFFER POINTER
        JMP     RXI1     ;GO BACK TO SEE IF MORE
RXI2:   SHLD    RCVPNT   ; (*) GET DATA BUFFER POINTER
        IN      RCVDAT   ; (*) READ DATA VIA RXDACK
        MOV     M,A      ; (*) STORE DATA IN BUFFER
        INX     H        ; (*) BUMP DATA POINTER
        JMP     RXI3     ; (*) DONE
RXI4:   MVI     A,01H    ;SET RX FLAG TO SHOW COMPLETION
        STA     RXFLAG   ;COMPLETION
RXI3:   POP     B        ;RESTORE BC
        POP     PSW      ;RESTORE PSW
        POP     H        ;RESTORE HL
        EI      ;ENABLE INTERRUPTS
        RET     ;DONE

;FUNCTION: TXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: TXRBUF, TXPNT, TXFLAG
;OUTPUTS: TXRBUF, TXPNT, TXFLAG
;CALLS: NONE
;DESTROYS: NOTHING
;DESCRIPTION: TXI IS ENTERED AT A TRANSMITTER INTERRUPT.
;THE INTERRUPT IS TESTED BY WAY OF THE IRA BIT TO SEE
;IF A DATA TRANSFER OR RESULT COMPLETION HAS OCCURED.
;FOR DATA TRANSFERS (IRA=0), THE DATA IS OBTAINED FROM
;A BUFFER LOCATION POINTED AT BY TXPNT.  FOR COMPLETION,
;(IRA=1), THE RESULTS ARE READ AND PLACED AT A RESULT
;BUFFER POINTED AT BY TXRBUF, AND THE TXFLAG IS SET
;TO INDICATE TO THE MAIN PROGRAM THAT A OPERATION IS
;COMPLETE. TX OPERATIONS HAVE ONLY ONE RESULT.
;DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*). THESE
;MAYBE REMOVED BY USERS USING DMA.
;
TXI:    PUSH    H        ;SAVE HL
        PUSH    PSW      ;SAVE PSW
        IN      STAT73   ; (*) READ 8273 STATUS
        ANI     TXIRA    ; (*) TEST TXIRA BIT
        JZ      TXI2     ; (*) IF 0, DATA TRANSFER
        IN      TXIR73   ;1, THEN READ TXIR
        LHLD    TXRBUF   ;GET RESULT BUFFER POINTER
        MOV     M,A      ;STORE RESULT IN BUFFER
        INX     H        ;BUMP RESULT POINTER
        SHLD    TXRBUF   ;RESTORE RESULT POINTER
        MVI     A,01H    ;SET TXFLAG TO SHOW COMPLETION
        STA     TXFLAG   ;SET FLAG
TXI1:   POP     PSW      ;RESTORE PSW
        POP     H        ;RESTORE HL
        EI      ;ENABLE INTERRUPTS
        RET     ;DONE
TXI2:   LHLD    TXPNT    ; (*) GET DATA POINTER
        MOV     A,M      ; (*) GET DATA FROM BUFFER
        OUT     TXDATA   ; (*) OUTPUT TO 8273 VIA TXDACK
        INX     H        ; (*) BUMP DATA POINTER
        SHLD    TXPNT    ; (*) RESTORE POINTER
        JMP     TXI1     ; (*) RETURN AFTER RESTORE
```

611001-61

Figure 22. Interrupt-Driven Result Handlers with Non-DMA Data Transfers

need the lines where the IRA bit is tested for zero. (These lines are denoted by an asterisk in the comments column.) Note that the INT bit is used to determine when all results have been read. All results must be read. Otherwise, the INT bit (and pin) will remain high and further interrupts may be missed. These routines place the results in a result buffer pointed at by RCRBUF and TxRBUF.

A typical result handler for systems utilizing polling is shown in Figure 23. Data transfers are also handled by this routine. This routine utilizes the routines of Figure 22 to handle the results.

At this point, the reader should have a good conceptual feel about how the 8273 operates. It is now time for the particulars of each command to be discussed.

```
;FUNCTION: POLOP
;INPUTS: NONE
;OUTPUTS: C=0 (NO STATUS), =1 (RX COMPLETION),
;   =2 (TX COMPLETION), =3 (BOTH)
;CALLS: TXI, RXI
;DESTROYS: B,C
;DESCRIPTION: POLOP IS CALLED TO POLL THE 8273 FOR
;DATA TRANSFERS AND COMPLETION RESULTS.  THE
;ROUTINES TXI AND RXI ARE USED FOR THE ACTUAL
;TRANSFERS AND BUFFER WORK.  POLOP RETURNS
;THE STATUS OF THEIR ACTION.
;
POLOP:  PUSH    PSW      ;SAVE PSW
        MVI     C,00H    ;CLEAR C
POLOP1: IN      STAT73   ;READ 8273 STATUS
        ANI     INT      ;ARE TXINT OR RXINT SET?
        JZ      PEXIT    ;NO, EXIT
        IN      STAT73   ;READ 8273 STATUS
        ANI     RXINT    ;TEST RX INT
        JNZ     RXIC     ;YES, GO SERVICE RX
        CALL    TXI      ;MUST BE TX, GO SERVICE IT
        LDA     TXFLAG   ;GET TX FLAG
        CPI     01H      ;WAS IT A COMPLETION? (01)
        JNZ     PEXIT    ;NO, SO JUST EXIT
        INR     C        ;YES, UPDATE C
        INR     C
        JMP     POLOP1   ;TRY AGAIN
;
RXIC:   CALL    RXI      ;GO SERVICE RX
        LDA     RXFLAG   ;GET RX FLAG
        CPI     01H      ;WAS IT A COMPLETION? (01)
        JNZ     PEXIT    ;NO, SO JUST EXIT
        INR     C        ;YES, UPDATE C
        JMP     POLOP1   ;TRY AGAIN
;
PEXIT:  POP     PSW      ;RESTORE PSW
        RET     ;RETURN WITH COMP. STATUS IN C
```

611001-62

Figure 23. Polling Result Handler

8273 COMMAND DESCRIPTION

In this section, each command is discussed in detail. In order to shorten the notation, please refer to the command key in Table 1. The 8273 utilizes five different command types: Initialization/Configuration, Receive, Transmit, Reset, and Modem Control.

Table 1. Command Summary Key

B_0, B_1	—LSB and MSB of Receive Buffer Length
R_0, R_1	—LSB and MSB of Received Frame Length
L_0, L_1	—LSB and MSB of Transmit Frame Length
A_1, A_2	—Match Addresses for Selective Receive
RIC	—Receiver Interrupt Result Code
TIC	—Transmitter Interrupt Result Code
A	—Address Field of Received Frame
C	—Control Field of Received Frame

Initialization/Configuration Commands

The Initialization/Configuration commands manipulate registers internal to the 8273 that define the various operating modes. These commands either set or reset specified bits in the registers depending on the type of command. One parameter is required. Set commands perform a logical OR operation of the parameter (mask) and the internal register. This mask contains 1s where register bits are to be set. A "0" in the mask causes no change in the corresponding register bit. Reset commands perform a logical AND operation of the parameter (mask) and the internal register, i.e., the mask is "0" to reset a register bit and a "1" to cause no change. Before presenting the commands, the register bit definitions are discussed.

Operating Mode Register (Figure 24)

D_7–D_6: *Not Used*—These bits must not be manipulated by any command; i.e., D_7–D_6 must be 0 for the Set command and 1 for the Reset command.

D_5: *HDLC Abort*—When this bit is set, the 8273 will interrupt when 7 1s (HDLC Abort) are received by an active receiver. When reset, an SDLC Abort (8 1s) will cause an interrupt.

D_4: *EOP Interrupt*—Reception of an EOP character (0 followed by 7 1s) will cause the 8273 to interrupt the CPU when this bit is set. Loop controller stations use this mode as a signal that a polling frame has completed the loop. No EOP interrupt is generated when this bit is reset.

D_3: *Early Tx Interrupt*—This bit specifies when the transmitter should generate an end of frame interrupt. If this bit is set, an interrupt is generated when the last data character has been passed to the 8273. If the user software issues another transmit command within two byte times, the final flag interrupt does not occur and the new frame is transmitted with only one flag of separation. If this restriction is not met, more than one flag will separate the frames and a frame complete interrupt is generated after the closing flag. If the bit is reset, only the frame complete interrupt occurs. This bit, when set, allows a single flag to separate consecutive frames.

D_2: *Buffered Address and Control*—When set, the address and control fields of received frames are buffered in the 8273 and passed to the CPU as results after a received frame interrupt (they are not transferred to memory with the information field). On transmit, the A and C fields are passed to the 8273 as parameters. This mode simplifies buffer management. When this bit is reset, the A and C fields are passed to and from memory as the first two data transfers.

D_1: *Preframe Sync*—When set, the 8273 prefaces each transmitted frame with two characters before the opening flag. These two characters provide 16 transitions to allow synchronization of the opposing receiver. To guarantee 16 transitions, the two characters are 55H-55H for non-NRZI mode (see Serial I/O Register description) or 00H-00H for NRZI mode. When reset, no preframe characters are transmitted.

D_0: *Flag Stream*—When set, the transmitter will start sending flag characters as soon as it is idle; i.e., immediately if idle when the command is issued or after a transmission if the transmitter is active when this bit is set. When reset, the transmitter starts sending Idle characters on the next character boundary if idle already, or at the end of a transmission if active.

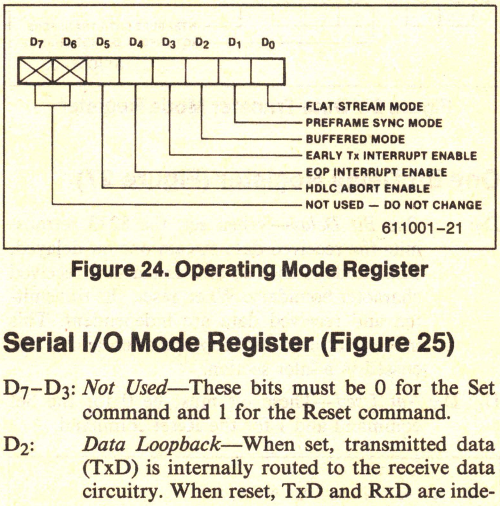


Figure 24. Operating Mode Register

Serial I/O Mode Register (Figure 25)

D_7–D_3: *Not Used*—These bits must be 0 for the Set command and 1 for the Reset command.

D_2: *Data Loopback*—When set, transmitted data (TxD) is internally routed to the receive data circuitry. When reset, TxD and RxD are independent.

D_1: *Clock Loopback*—When set, $\overline{\text{TxC}}$ is internally routed to $\overline{\text{RxC}}$. When reset, the clocks are independent.

D_0: *NRZI (Non-Return to Zero Inverted*—When set, the 8273 assumes the received data is NRZI encoded, and NRZI encodes the transmitted data. When reset, the received and transmitted data are treated as a normal positive logic bit stream.

Data Transfer Mode Register (Figure 26)

D_7–D_1: *Not Used*—These bits must be 0 for the Set command and 1 for the Reset command.

D_0: *Interrupt Data Transfer*—When set, the 8273 will interrupt the CPU when data transfers are required (the corresponding IRA Status register bit will be 0 to signify a data transfer interrupt rather than a Result phase interrupt). When reset, 8273 data transfers are performed through DMA requests on the DRQ pins without interrupting the CPU.

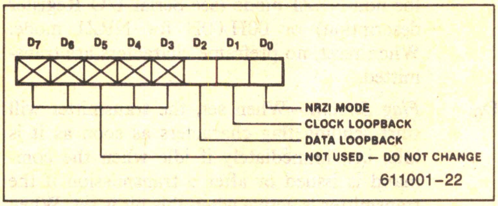


Figure 25. Serial I/O Mode Register

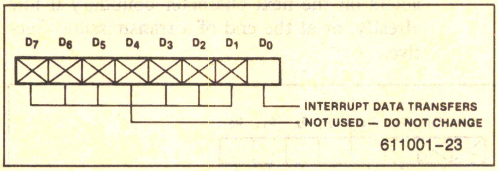


Figure 26. Data Transfer Mode Register

One Bit Delay Register (Figure 27)

D_7: *One Bit Delay*—When set, the 8273 retransmits the received data stream one bit delayed. This mode is entered and exited at a received character boundary. When reset, the transmitted and received data are independent. This mode is utilized for loop operation and is discussed in a later section.

D_6–D_0: *Not Used*—These bit must be 0 for the Set command and 1 for the Reset command.

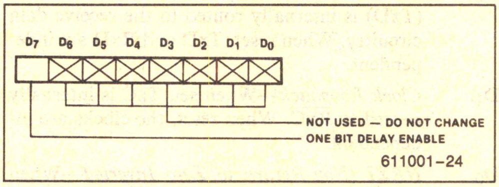


Figure 27. One Bit Delay Mode Register

Figure 28 shows the Set and Reset commands associated with the above registers. The mask which sets or resets the desired bits is treated as a single parameter. These commands do not interrupt nor provide results during the Result phase. After reset, the 8273 defaults to all of these bits reset.

Register	Command	Hex Code	Parameter
One Bit Delay Mode	Set	A4	Set Mask
	Reset	64	Reset Mask
Data Transfer Mode	Set	97	Set Mask
	Reset	57	Reset Mask
Operating Mode	Set	91	Set Mask
	Reset	51	Reset Mask
Serial I/O Mode	Set	A0	Set Mask
	Reset	60	Reset Mask

Figure 28. Initialization/Configuration Command Summary

Receive Commands

The 8273 supports three receive commands plus a receiver disable function.

General Receive

When commanded to General Receive, the 8273 passes all frames either to memory (DMA mode) or to the CPU (non-DMA mode) regardless of the contents of the frame's address field. This command is used for primary and loop controller stations. Two parameters are required: B_0 and B_1. These parameters are the LSB and MSB of the receiver buffer size. Giving the 8273 this extra information alleviates the CPU of the burden of checking for buffer overflow. The 8273 will interrupt the CPU if the received frame attempts to overfill the allotted buffer space.

Selective Receive

In Selective Receive, two additional parameters besides B_0 and B_1 are required: A_1 and A_2. These parameters are two address match bytes. When commanded to Selective Receive, the 8273 passes to memory or the CPU only those frames having an address field matching either A_1 or A_2. This command is usually used for secondary stations with A_1 being the secondary address and A_2 is the "All Parties" address. If only one match byte is needed, A_1 and A_2 should be equal. As in General Receive, the 8273 counts the incoming data bytes and interrupts the CPU if B_0, B_1 is exceeded.

Selective Loop Receive

This command is very similar in operation to Selective Receive except that One Bit Delay mode must be set

and that the loop is captured by placing transmitter in Flag Stream mode automatically after an EOP character is detected following a selectively received frame. The details of using the 8273 in loop configurations is discussed in a later section so please hold questions until then.

The handling of interrupt results is common among the three commands. When a frame is received without error, i.e., the FCS is correct and $\overline{CD}$ (Carrier Detect) was active throughout the frame or no attempt was made to overfill the buffer; the 8273 interrupts the CPU following the closing flag to pass the completion results. These results, in order, are the receiver interrupt result code (RIC), and the byte length of the information field of the received frame (R_0, R_1). If Buffered mode is selected, the address and control fields are passed as two additional results. If Buffered mode is not selected, the address and control fields are passed as the first two data transfers and R_0, R_1 reflect the information field length plus two.

Receive Disable

The receiver may also be disabled using the Receive Disable command. This command terminates any receive operation immediately. No parameters are required and no results are returned.

The details for the Receive command are shown in Figure 29. The interrupt result code key is shown in Figure 30. Some explanation of these result codes is appropriate.

The interrupt result code is the first byte passed to the CPU in the RxI/R register during the Result phase. Bits D_4-D_0 define the cause of the receiver interrupt. Since each result code has specific implications, they are discussed separately below.

Command	Hex Code	Parameters	Results* RxI/R
General Receive	C0	B_0, B_1	RIC, R_0, R_1, A, C
Selective Receive	C1	B_0, B_1, A_1, A_2	RIC, R_0, R_1, A, C
Selective Loop Receive	C2	B_0, B_1, A_1, A_2	RIC, R_0, R_1, A, C
Disable Receiver	C5	None	None

***NOTE:**
A and C are passed as results only in buffered mode.

Figure 29. Receiver Command Summary

RIC D_7-D_0	Receiver Interrupt Result Code	Rx Status After INT
* 00000	A_1 Match or General Receive	Active
* 00001	A_2 Match	Active
000 00011	CRC Error	Active
000 00100	Abort Detected	Active
000 00101	Idle Detected	Disabled
000 00110	EOP Detected	Disabled
000 00111	Frame < 32 Bits	Active
000 01000	DMA Overrun	Disabled
000 01001	Memory Buffer Overflow	Disabled
000 01010	Carrier Detect Failure	Disabled
000 01011	Receiver Interrupt Overrun	Disabled
***D_7-D_5**	**Partial Byte Received**	
111	All 8 Bits of Last Byte	
000	D_0	
100	D_1-D_0	
010	D_2-D_0	
110	D_3-D_0	
001	D_4-D_0	
101	D_5-D_0	
011	D_6-D_0	

Figure 30. Receiver Interrupt Result Codes (RIC)

The first two result codes result from the error-free reception of a frame. If the frame is received correctly after a General Receive command, the first result is returned. If either Selective Receive command was used (normal or loop), a match with A_1 generates the first result code and a match with A_2 generates the second. In either case, the receiver remains active after the interrupt; however, the internal buffer size counters are not reset. That is, if the receive command indicated 100 bytes were allocated to the receive buffer (B_0, B_1) and an 80-byte frame was received correctly, the maximum next frame size that could be received without recommanding the receiver (resetting B_0 and B_1) is 20 bytes. Thus, it is common practice to recommand the receiver after each frame reception. DMA and/or memory pointers are usually updated at this time. (Note that users who do not wish to take advantage of the 8273's buffer management features may simply use B_0, B_1 = 0FFH for each receive command. Then frames of 65K bytes may be received without buffer overflow errors.)

The third result code is a CRC error. This indicates that a frame was received in the correct format (flags, etc.); however, the received FCS did not check with the internally generated FCS. The frame should be discarded. The receiver remains active. (Do not forget that even though an error condition has been detected, all frame information up until that error has either been transferred to memory or passed to the CPU. This information should be invalidated. This applies to all receiver error conditions.) Note that the FCS, either transmitted or received, is never available to the CPU.

The Abort Detect result occurs whenever the receiver sees either an SDLC (8 1s) or an HDLC (7 1s), depending on the Operating Mode register. However, the intervening Abort character between a closing flag and an Idle does not generate an interrupt. If an Abort character (seen by an active receiver within a frame) is not preceded by a flag and is followed by an idle, an interrupt will be generated for the Abort, followed by an Idle interrupt one character time later. The Idle Detect result occurs whenever 15 consecutive 1s are received. After the Abort Detect interrupt, the receiver remains active. After the Idle Detect interrupt, the receiver is disabled and must be recommanded before further frames may be received.

If the EOP Interrupt bit is set in the Operating Mode register, the EOP Detect result is returned whenever an EOP character is received. The receiver is disabled, so the Idle following the EOP does not generate an Idle Detect interrupt.

The minimum number of bits in a valid frame between the flags is 32. Fewer than 32 bits indicates an error. If Buffered mode is selected, such frames are ignored, i.e., no data transfers or interrupts are generated. In non-Buffered mode, a < 32-bit frame generates an interrupt with the < 32-bit frame result since data transfers may already have disturbed the 8257 or interrupt handler. The receiver remains active.

The DMA Overrun results from the $\overline{\text{DMA}}$ controller being too slow in extracting data from the 8273, i.e., the $\overline{\text{RxDACK}}$ signal is not returned before the next received byte is ready for transfer. The receiver is disabled if this error condition occurs.

The Memory Buffer Overflow result occurs when the number of received bytes exceeds the receiver buffer length supplied by the B_0 and B_1 parameters in the receive command. The receiver is disabled.

The Carrier Detect Failure result occurs when the $\overline{\text{CD}}$ pin goes high (inactive) during reception of a frame. The $\overline{\text{CD}}$ pin is used to qualify reception and must be active by the time the address field starts to be received. If $\overline{\text{CD}}$ is lost during the frame, a $\overline{\text{CD}}$ Failure interrupt is generated and the receiver is disabled. No interrupt is generated if $\overline{\text{CD}}$ goes inactive between frames.

If a condition occurs requiring an interrupt be generated before the CPU has finished reading the previous interrupt results, the second interrupt is generated after the current Result phase is complete (the RxINT pin and status bit go low then high). However, the interrupt result for this second interrupt will be a Receive Interrupt Overrun. The actual cause of the second interrupt is lost. One case where this may occur is at the end of a received frame where the line goes idle. The 8273 generates a received frame interrupt after the closing flag and then 15-bit times later, generates an Idle Detect interrupt. If the interrupt service routine is slow in reading the first interrupt's results, the internal RxI/R register still contains result information when the Idle Detect interrupt occurs. Rather than wiping out the previous results, the 8273 adds a Receive Interrupt Overrun result as an extra result. If the system's interrupt structure is such that the second interrupt is not acknowledged (interrupts are still disabled from the first interrupt), the Receive Interrupt Overrun result is read as an extra result, after those from the first interrupt. If the second interrupt is serviced, the Receive Interrupt Overrun is returned as a single result. (Note that the INT pins supply the necessary transitions to support a Programmable Interrupt Controller such as the Intel 8259. Each interrupt generates a positive-going edge on the appropriate INT pin and the high level is held until the interrupt is completely serviced.) In general, it is possible to have interrupts occurring at one character time intervals. Thus the interrupt handling software must have at least that much response and service time.

The occurrence of Receive Interrupt Overruns is an indication of marginal software design; the system's interrupt response and servicing time is not sufficient for the

data rates being attempted. It is advisable to configure the interrupt handling software to simply read the interrupt results, place them into a buffer, and clear the interrupt as quickly as possible. The software can then examine the buffer for new results at its leisure, and take appropriate action. This can easily be accomplished by using a result buffer flag that indicates when new results are available. The interrupt handler sets the flag and the main program resets it once the results are retrieved.

Both SDLC and HDLC allow frames which are of arbitrary length (>32 bits). The 8273 handles this N-bit reception through the high order bits (D_7–D_5) of the result code. These bits code the number of valid received bits in the last received information field byte. This coding is shown in Figure 30. The high order bits of the received partial byte are indeterminate. [The address, control, and information fields are transmitted least significant bit (A_0) first. The FCS is complemented and transmitted most significant bit first.]

Transmit Commands

The 8273 transmitter is supported by three Transmit commands and three corresponding Abort commands.

Transmit Frame

The Transmit Frame command simply transmits a frame. Four parameters are required when Buffered mode is selected and two when it is not. In either case, the first two parameters are the least and the most significant bytes of the desired frame length (L_0, L_1). In Buffered mode, L_0 and L_1 equal the length in bytes of the desired information field, while in the non-Buffered mode, L_0 and L_1 must be specified at the information field length plus two. (L_0 and L_1 specify the number of data transfers to be performed.) In Buffered mode, the address and control fields are presented to the transmitter as the third and fourth parameters respectively. In non-Buffered mode, the A and C fields must be passed as the first two data transfers.

When the Transmit Frame command is issued, the 8273 makes $\overline{RTS}$ (Request-to-Send) active (pin low) if it was not already. It then waits until $\overline{CTS}$ (Clear-to-Send) goes active (pin low) before starting the frame. If the Preframe Sync bit in the Operating Mode register is set, the transmitter prefaces two characters (16 transitions) before the opening flag. If the Flag Stream bit is set in the Operating Mode register, the frame (including Preframe Sync if selected) is started on a flag boundary. Otherwise the frame starts on a character boundary.

At the end of the frame, the transmitter interrupts the CPU (the interrupt results are discussed shortly) and returns to either Idle or Flag Stream, depending on the Flag Stream bit of the Operating Mode register. If $\overline{RTS}$ was active before the transmit command, the 8273 does not change it. If it was inactive, the 8273 will deactivate it within one character time.

Loop Transmit

Loop Transmit is similar to Frame Transmit (the parameter definition is the same). But since it deals with loop configurations, One Bit Delay mode must be selected.

If the transmitter is not in Flag Stream mode when this command is issued, the transmitter waits until after a received EOP character has been converted to a flag (this is done automatically) before transmitting. (The one bit delay is, of course, suspended during transmit.) If the transmitter is already in Flag Stream mode as a result of a selectively received frame during a Selective Loop Receive command, transmission will begin at the next flag boundary for Buffered mode or at the third flag boundary for non-Buffered mode. This discrepancy is to allow time for enough data transfers to occur to fill up the internal transmit buffer. At the end of a Loop Transmit, the One Bit Delay mode is re-entered and the flag stream mode is reset. More detailed loop operation is covered later.

Transmit Transparent

The Transmit Transparent command enables the 8273 to transmit a block of raw data. This data is without SDLC protocol, i.e., no zero bit insertion, flags, or FCS. Thus it is possible to construct and transmit a Bi-Sync message for front-end processor switching or to construct and transmit an SDLC message with incorrect FCS for diagnostic purposes. Only the L_0 and L_1 parameters are used since there are not fields in this mode. (The 8273 does not support a Receive Transparent command.)

Abort Commands

Each of the above transmit commands has an associated Abort command. The Abort Frame Transmit command causes the transmitter to send eight contiguous ones (no zero bit insertion) immediately and then revert to either idle or flag streaming based on the Flag Stream bit. (The 8 1s as an Abort character is compatible with both SDLC and HDLC.)

For Loop Transmit, the Abort Loop Transmit command causes the transmitter to send one flag and then revert to one bit delay. Loop protocol depends upon FCS errors to detect aborted frames.

The Abort Transmit Transparent simply causes the transmitter to revert to either idles or flags as a function of the Flag Stream mode specified.

The Abort commands require no parameters, however, they do generate an interrupt and return a result when complete.

A summary of the Transmit commands is shown in Figure 31. Figure 32 shows the various transmit interrupt result codes. As in the receiver operation, the transmitter generates interrupts based on either good completion of an operation or an error condition to start the Result phase.

The Early Transmit Interrupt result occurs after the last data transfer to the 8273 if the Early Transmit Interrupt bit is set in the Operating Mode register. If the 8273 is commanded to transmit again within two character times, a single flag will separate the frames. (Buffered mode must be used for a single flag to separate the frames. If non-Buffered mode is selected, three flags will separate the frames.) If this time constraint is not met, another interrupt is generated and multiple flags or idles will separate the frames. The second interrupt is the normal Frame Transmit Complete interrupt. The Frame Transmit Complete result occurs at the closing flag to signify a good completion.

The DMA Underrun result is analogous to the DMA Overrun result in the receiver. Since SDLC does not support intraframe time fill, if the DMA controller or CPU does not supply the data in time, the frame must be aborted. The action taken by the transmitter on this error is automatic. It aborts the frame just as if an Abort command had been issued.

Clear-to-Send Error result is generated if $\overline{CTS}$ goes inactive during a frame transmission. The frame is aborted as above.

The Abort Complete result is self-explanatory. Please note however that no Abort Complete interrupt is generated when an automatic abort occurs. The next command type consists of only one command.

Reset Command

The Reset command provides a software reset function for the 8273. It is a special case and does not utilize the normal command interface. The reset facility is provided in the Test Mode register. The 8273 is reset by simply outputting a 01H followed by a 00H to the Test Mode register. Writing the 01 followed by the 00 mimicks the action required by the hardware reset. Since the 8273 requires time to process the reset internally, at least 10 cycles of the ϕCLK clock must occur between the writing of the 01 and the 00. The action taken is the same as if a hardware reset is performed, namely:

1) The modem control outputs are forced high inactive.

Command	Hex Code	Parameters*	Results TxI/R
Transmit Frame	C8	L_0, L_1, A, C	TIC
Abort	CC	None	TIC
Loop Transmit	CA	L_0, L_1, A, C	TIC
Abort	CE	None	TIC
Transmit Transparent	C0	L_0, L_1	TIC
Abort	CD	None	TIC

***NOTE:**
A and C are passed as parameters in buffered mode only.

Figure 31. Transmitter Command Summary

RIC D_7–D_0	Transmitter Interrupt Result Code	Tx Status after INT
000 01100	Early Tx Interrupt	Active
000 01101	Frame Tx Complete	Idle or Flags
000 01110	DMA Underrun	Abort
000 01111	Clear to Send Error	Abort
000 10000	Abort Complete	Idle or Flags

Figure 32. Transmitter Interrupt Result Codes

2) The 8273 Status register is cleared.

3) Any commands in progress cease.

4) The 8273 enters an idle state until the next command is issued.

Modem Control Commands

The modem control ports were discussed earlier in the Hardware section. The commands used to manipulate these ports are shown in Figure 33. The Read Port A and Read Port B commands are immediate. The bit definition for the returned byte is shown in Figures 13 and 14. Do not forget that the returned value represents the logical condition of the pin, i.e., pin active (low) = bit set.

The Set and Reset Port B commands are similar to the Initialization commands in that they use a mask parameter which defines the bits to be changed. Set Port B utilizes a logical OR mask and Reset Port B uses a logical AND mask. Setting a bit makes the pin active (low). Resetting the bit deactivates the pin (high).

To help clarify the numerous timing relationships that occur and their consequences, Figures 34 and 35 are provided as an illustration of several typical sequences. It is suggested that the reader go over these diagrams and re-read the appropriate part of the previous sections if necessary.

HDLC CONSIDERATIONS

The 8273 supports HDLC as well as SDLC. Let's discuss how the 8273 handles the three basic HDLC/SDLC differences: extended addressing, extended control, and the 7 1s Abort character.

Recalling Figure 4a, HDLC supports an address field of indefinite length. The actual amount of extension used is determined by the least significant bit of the characters immediately following the opening flag. If the LSB is 0, more address field bytes follow. If the LSB is 1, this byte is the final address field byte. Software must be used to determine this extension.

If non-Buffered mode is used, the A, C, and I fields are in memory. The software must examine the initial characters to find the extent of the address field. If Buffered mode is used, the characters corresponding to the SDLC A and C fields are transferred to the CPU as interrupt results. Buffered mode assumes the two characters following the opening flag are to be transferred as interrupt results regardless of content or meaning. (The 8273 does not know whether it is being used in an SDLC or an HDLC environment.) In SDLC, these characters are necessarily the A and C field bytes, however in HDLC, their meaning may change depending on the amount of extension used. The software must recognize this and examine the transferred results as possible address field extensions.

Frames may still be selectively received as is needed for secondary stations. The Selective Receive command is still used. This command qualifies a frame reception on the first byte following the opening flag matching either of the A_1 or A_2 match byte parameters. While this does not allow qualification over the complete range of HDLC addresses, it does perform a qualification on the first address byte. The remaining address field bytes, if any, are then examined via software to completely qualify the frame.

Once the extent of the address field is found, the following bytes form the control field. The same LSB test used for the address field is applied to these bytes to determine the control field extension, up to two bytes maximum. The remaining frame bytes in memory represent the information field.

The Abort character difference is handled in the Operating Mode register. If the HDLC Abort Enable bit is set, the reception of seven contiguous ones by an active receiver will generate an Abort Detect interrupt rather than eight ones. (Note that both the HDLC Abort Enable bit and the EOP Interrupt bit must not be set simultaneously.)

Now let's move on to the SDLC loop configuration discussion.

Port	Command	Hex Code	Parameter	Reg Result
A Input	Read	22	None	Port Value
B Output	Read	23	None	Port Value
	Set	A3	Set Mask	None
	Reset	63	Reset Mask	None

Figure 33. Modem Control Command Summary

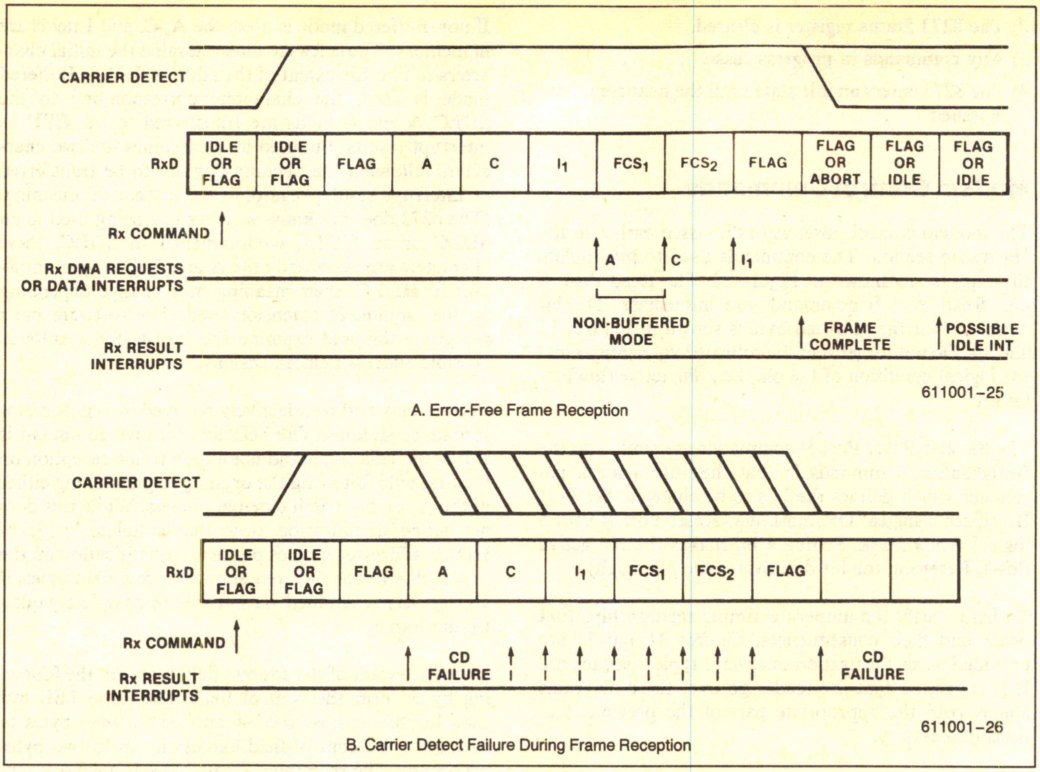


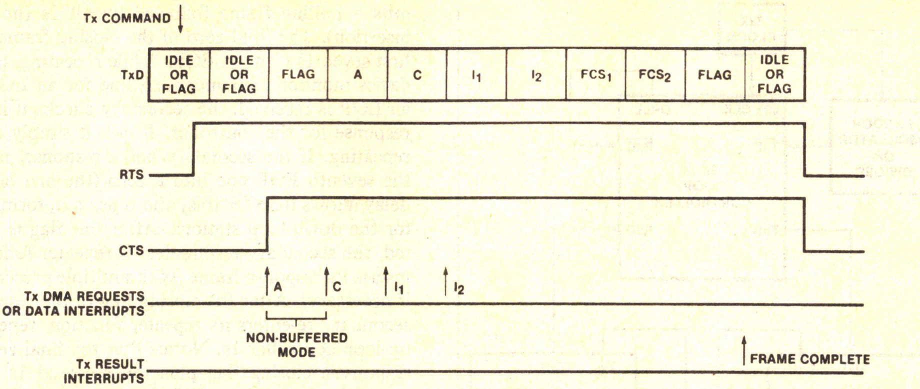
Figure 34. Sample Receiver Timing Diagrams

LOOP CONFIGURATION

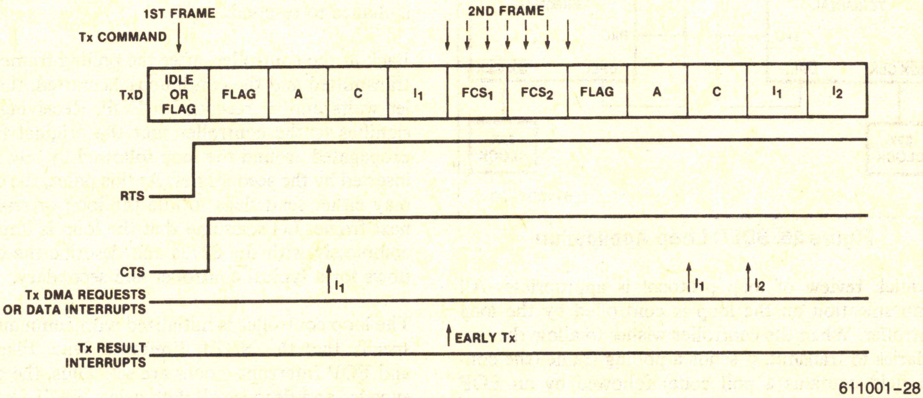
Aside from use in the normal data link applications, the 8273 is extremely attractive in loop configuration due to the special frame-level loop commands and the Digital Phase Locked Loop. Toward this end, this section details the hardware and software considerations when using the 8273 in a loop application.

The loop configuration offers a simple, low-cost solution for systems with multiple stations within a small physical location, i.e., retail stores and banks. There are two primary reasons to consider a loop configuration. The interconnect cost is lower for a loop over a multi-point configuration since only one twisted pair or fiber optic cable is used. (The loop configuration does not support the passing of distinct clock signals from station to station.) In addition, loop stations do not need the intelligence of a multi-point station since the loop protocol is simpler. The most difficult aspects of loop station design are clock recovery and implementation of one bit delay (both are handled neatly by the 8273).

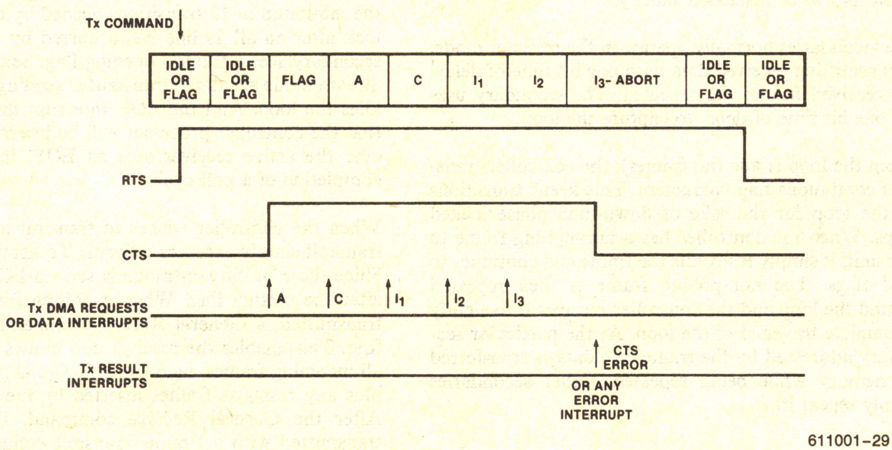
Figure 36 illustrates a typical loop configuration with one controller and two down-loop secondaries. Each station must derive its own data timing from the received data stream. Recalling our earlier discussion of the DPLL, notice that $\overline{TxC}$ and $\overline{RxC}$ clocks are provided by the $\overline{DPLL}$ output. The only clock required in the secondaries is a simple, non-synchronized clock at 32 times the desired baud rate. The controller requires both 32× and 1× clocks. (The 1× is usually implemented by dividing the 32× clock with a 5-bit divider. However, there is no synchronism requirement between these clocks so any convenient implementation may be used.)



A. Error-Free Frame Transmission



B. Diagram Showing Tx Command Queing and Early Tx Interrupt
(Single flag between frames) Buffered Mode is Assumed



C. CTS Failure (or other error) During Transmission

Figure 35. Sample Transmitter Timing Diagrams

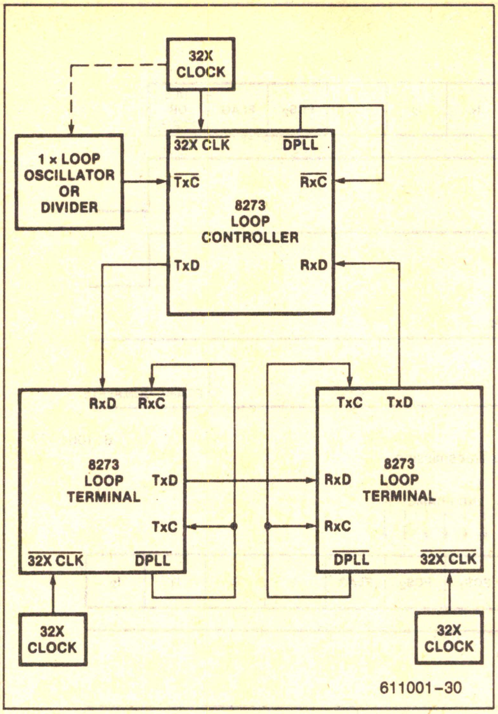


Figure 36. SDLC Loop Application

A quick review of loop protocol is appropriate. All communication on the loop is controlled by the loop controller. When the controller wishes to allow the secondaries to transmit, it sends a polling frame (the control field contains a poll code) followed by an EOP (End-of-Poll) character. The secondaries use the EOP character to capture the loop and insert a response frame as will be discussed shortly.

The secondaries normally operate in the repeater mode, retransmitting received data with one bit time of delay. All received frames are repeated. The secondary uses the one bit time of delay to capture the loop.

When the loop is idle (no frames), the controller transmits continuous flag characters. This keeps transitions on the loop for the sake of down-loop phase locked loops. When the controller has a non-polling frame to transmit, it simply transmits the frame and continues to send flags. The non-polling frame is then repeated around the loop and the controller receives it to signify a complete traversal of the loop. At the particular secondary addressed by the frame, the data is transferred to memory while being repeated. Other secondaries simply repeat it.

If the controller wants to poll the secondaries, it transmits a polling frame followed by all 1s (no zero bit insertion). The final zero of the closing frame plus the first seven 1s form an EOP. While repeating, the secondaries monitor their incoming line for an EOP. When an EOP is received, the secondary checks if it has any response for the controller. If not, it simply continues repeating. If the secondary has a response, it changes the seventh EOP one into a zero (the one bit time of delay allows time for this) and repeats it, forming a flag for the down-loop stations. After this flag is transmitted, the secondary terminates its repeater function and inserts its response frame (with multiple preceding flags if necessary). After the closing flag of the response, the secondary re-enters its repeater function, repeating the up-loop controller 1s. Notice that the final zero of the response's closing flag plus the repeated 1s from the controller form a new EOP for the next down-loop secondary. This new EOP allows the next secondary to insert a response if it desires. This gives each secondary a chance to respond.

Back at the controller, after the polling frame has been transmitted and the continuous 1s started, the controller waits until it receives an EOP. Receiving an EOP signifies to the controller that the original frame has propagated around the loop followed by any responses inserted by the secondaries. At this point, the controller may either send flags to idle the loop or transmit the next frame. Let's assume that the loop is implemented completely with the 8273s and describe the command flows for a typical controller and secondary.

The loop controller is initialized with commands which specify that the NRZI, Preframe Sync, Flag Stream, and EOP Interrupt modes are set. Thus, the controller encodes and decodes all data using NRZI format. Preframe Sync mode specifies that all transmitted frames be prefaced with 16 line transitions. This ensures that the minimum of 12 transitions needed by the DPLL to lock after an all 1s line has occurred by the time the secondary sees a frame's opening flag. Setting the Flag Stream mode starts the transmitter sending flags which idles the loop. And the EOP Interrupt mode specifies that the controller processor will be interrupted whenever the active receiver sees an EOP, indicating the completion of a poll cycle.

When the controller wishes to transmit a non-polling frame, it simply executes a Frame Transmit command. Since the Flag Stream mode is set, no EOP is formed after the closing flag. When a polling frame is to be transmitted, a General Receive command is executed first. This enables the receiver and allows reception of all incoming frames; namely, the original polling frame plus any response frames inserted by the secondaries. After the General Receive command, the frame is transmitted with a Frame Transmit command. When the frame is complete, a transmitter interrupt is gener-

ated. The loop controller processor uses this interrupt to reset Flag Stream mode. This causes the transmitter to start sending all 1s. An EOP is formed by the last flag and the first 7 1s. This completes the loop controller transmit sequence.

At any time following the start of the polling frame transmission the loop controller receiver will start receiving frames. (The exact time difference depends, of course, on the number of down-loop secondaries due to each inserting one bit time of delay.) The first received frame is simply the original polling frame. However, any additional frames are those inserted by the secondaries. The loop controller processor knows all frames have been received when it sees an EOP Interrupt. This interrupt is generated by the 8273 since the EOP Interrupt mode was set during initialization. At this point, the transmitter may be commanded either to enter Flag Stream mode, idling the loop, or to transmit the next frame. A flowchart of this sequence is shown in Figure 37.

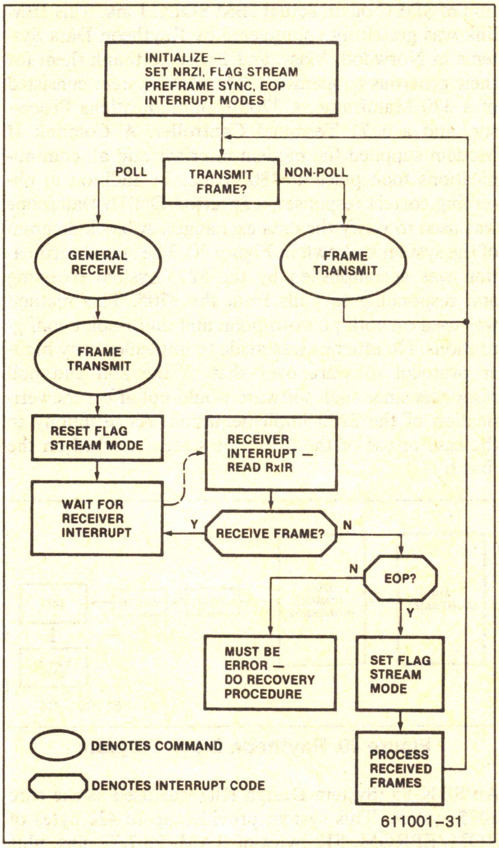


Figure 37. Loop Controller Flowchart

The secondaries are initialized with the NRZI and One Bit Delay modes set. This puts the 8273 into the repeater mode with the transmitter repeating the received data with one bit time of delay. Since a loop station cannot transmit until it sees an EOP character, any transmit command is queued until an EOP is received. Thus whenever the secondary wishes to transmit a response, a Loop Transmit command is issued. The 8273 then waits until it receives an EOP. At this point, the receiver changes the EOP into a flag, repeats it, resets One Bit Delay mode stopping the repeater function, and sets the transmitter into Flag Stream mode. This captures the loop. The transmitter now inserts its message. At the closing flag, Flag Stream mode is reset, and One Bit Delay mode is set, returning the 8273 to repeater function and forming an EOP for the next down-loop station. These actions happen automatically after a Loop Transmit command is issued.

When the secondary wants its receiver enabled, a Selective Loop Receive command is issued. The receiver then looks for a frame having a match in the Address field. Once such a frame is received, repeated, and transferred to memory, the secondary's processor is interrupted with the appropriate Match interrupt result and the 8273 continues with the repeater function until an EOP is received, at which point the loop is captured as above. The processor should use the interrupt to determine if it has a message for the controller. If it does, it simply issues a Loop Transmit command and things progress as above. If the processor has no message, the software must reset the Flag Stream mode bit in the Operating Mode register. This will inhibit the 8273 from capturing the loop at the EOP. (The match frame and the EOP may be separated in time by several frames depending on how many up-loop stations inserted messages of their own.) If the timing is such that the receiver has already captured the loop when the Flag Stream mode bit is reset, the mode is exited on a flag boundary and the frame just appears to have extra closing flags before the EOP. Notice that the 8273 handles the queuing of the transmit commands and the setting and resetting of the mode bits automatically. Figure 38 illustrates the major points of the secondary command sequence.

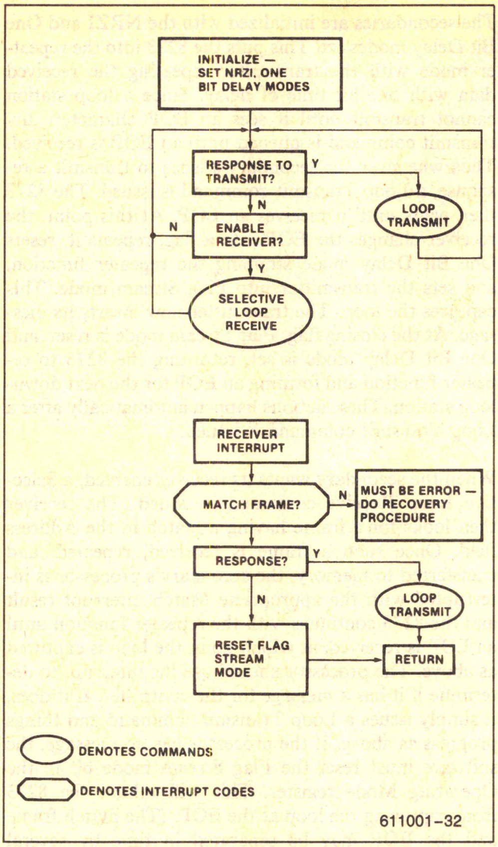


Figure 38. Loop Secondary Flowchart

When an off-line secondary wishes to come on-line, it must do so in a manner which does not disturb data on the loop. Figure 39 shows a typical hardware interface. The line labeled Port could be one of the 8273 Port B outputs and is assumed to be high (1) initially. Thus up-loop data is simply passed down-loop with no delay; however, the receiver may still monitor data on the loop. To come on-line, the secondary is initialized with only the EOP Interrupt mode set. The up-loop data is then monitored until an EOP occurs. At this point, the secondary's CPU is interrupted with an EOP interrupt. This signals the CPU to set One Bit Delay mode in the 8273 and then to set Port low (active). These actions switch the secondary's one bit delay into the loop. Since after the EOP only 1s are traversing the loop, no loop disturbance occurs. The secondary now waits for the next EOP, captures the loop, and inserts a "new on-line" message. This signals the controller that a new secondary exists and must be acknowledged. After the secondary receives its acknowledgement, the normal command flow is used.

It is hopefully evident from the above discussion that the 8273 offers a very simple and easy to implement solution for designing loop stations whether they are controllers or down-loop secondaries.

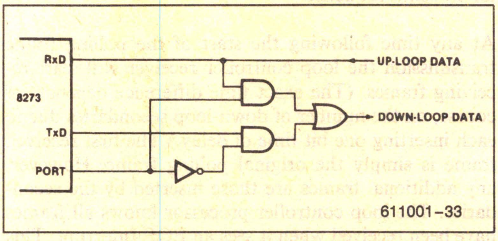


Figure 39. Loop Interface

APPLICATION EXAMPLE

This section describes the hardware and software of the 8273/8085 system used to verify the 8273 implementation of SDLC on an actual IBM SDLC Link. This IBM link was gratefully volunteered by Raytheon Data Systems in Norwood, Mass. and I wish to thank them for their generous cooperation. The IBM system consisted of a 370 Mainframe, a 3705 Communications Processor, and a 3271 Terminal Controller. A Comlink II Modem supplied the modem interface and all communications took place at 4800 baud. In addition to observing correct responses, a Spectron D601B Datascope was used to verify the data exchanges. A block diagram of the system is shown in Figure 40. The actual verification was accomplished by the 8273 system receiving and responding to polls from the 3705. This method was used on both point-to-point and multi-point configurations. No attempt was made to implement any higher protocol software over that of the poll and poll responses since such software would not affect the verification of the 8273 implementation. As testimony to the ease of use of the 8273, the system worked on the first try.

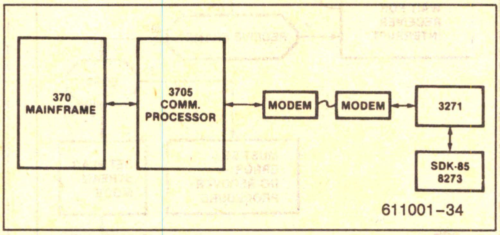


Figure 40. Raytheon Block Diagram

An SDK-85 (System Design Kit) was used as the core 8085 system. This system provides up to 4K bytes of ROM/EPROM, 512 bytes of RAM, 76 I/O pins, plus

two timers as provided in two 8755 Combination EPROM/I/O devices and two 8155 Combination RAM/I/O/Timer devices. In addition, 5 interrupt inputs are supplied on the 8085. The address, data, and control buses are buffered by the 8212 and 8216 latches and bidirectional bus drivers. Although it was not used in this application, an 8279 Display Driver/Keyboard Encoder is included to interface the on-board display and keyboard. A block diagram of the SDK-85 is shown in Figure 41. The 8273 and associated circuitry was constructed on the ample wire-wrap area provided for the user.

The example 8237/8085 system is interrupt-driven and uses DMA for all data transfers supervised by an 8257 DMA Controller. A 2400 baud asynchronous line, implemented with an 8251A USART, provides communication between the software and the user. 8253 Programmable Interval Timer is used to supply the baud rate clocks for the 8251A and 8273. (The 8273 baud rate clocks were used only during initial system debug. In actual operation, the modem supplied these clocks via the RS-232 interface.) Two 2142 1K x 4 RAMs provided 512 bytes of transmitter and 512 bytes of receiver buffer memory. (Command and result buffers, plus miscellaneous variables are stored in the 8155s.) The RS-232 interface utilized MC1488 and MC1489 RS-232 drivers and receivers. The schematic of the system is shown in Figure 42.

One detail to note is the DMA and interrupt structure of the transmit and receive channels. In both cases, the receiver is always given the higher priority (8257 DMA channel 0 has priority over the remaining channels and the 8085 RST 7.5 interrupt input has priority over the RST 6.5 input.) Although the choice is arbitrary, this technique minimizes the chance that received data could be lost due to other processor or DMA commitments.

Also note that only one 8205 Decoder is used for both peripheral and memory Chip Select. This was done to eliminate separate memory and I/O decoders since it was known beforehand that neither address space would be completely filled.

The 4 MHz crystal and 8224 Clock Generator were used only to verify that the 8273 operates correctly at that maximum spec speed. In a normal system, the 3.072 MHz clock from the 8085 would be sufficient. (This fact was verified during initial checkout.)

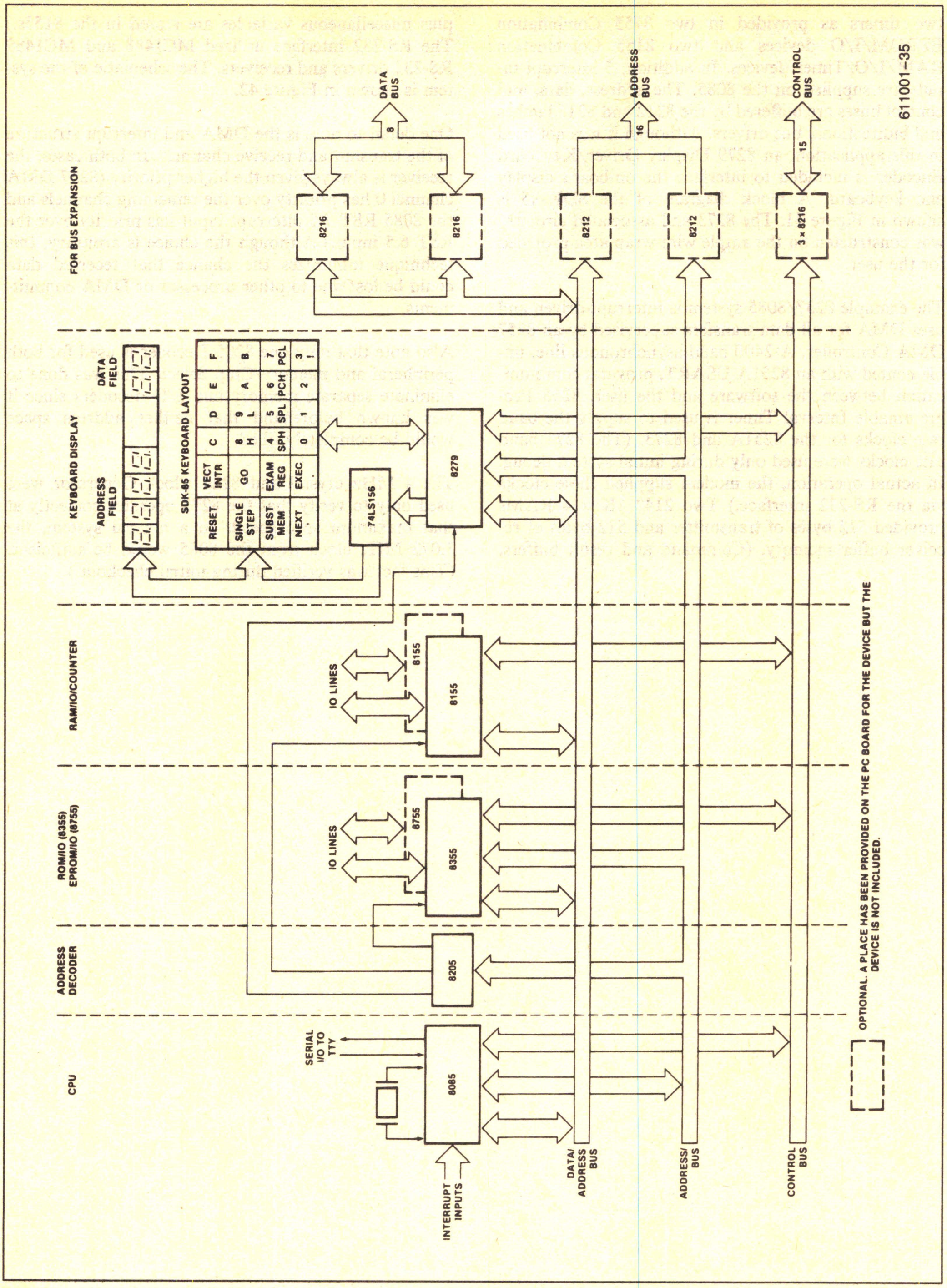


Figure 41. SDK-85 Functional Block Diagram

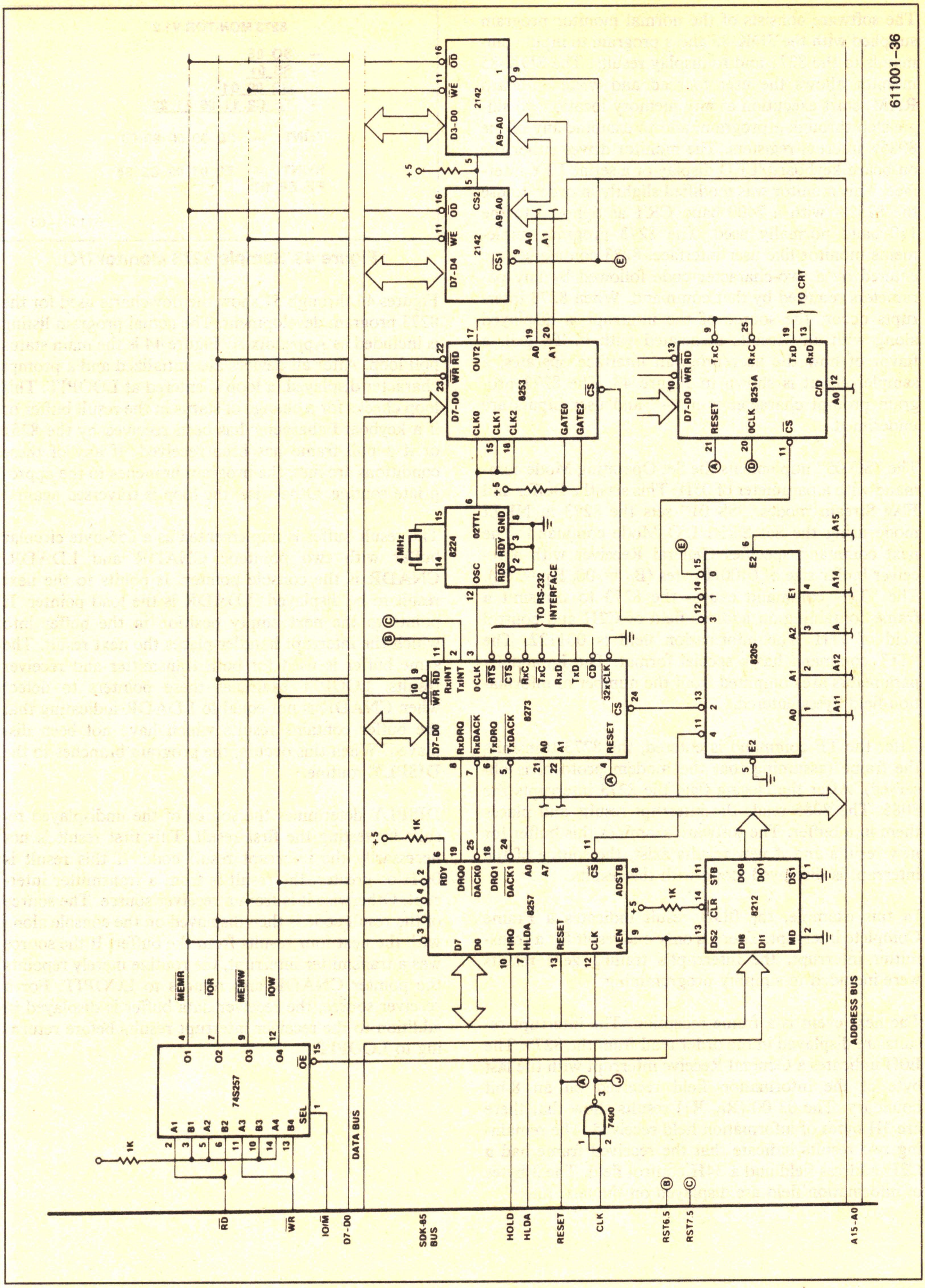


Figure 42. 8273/SDK-85 System

The software consists of the normal monitor program supplied with the SDK-85 and a program to input commands to the 8273 and to display results. The SDK-85 monitor allows the user to read and write on-board RAM, start execution at any memory location, to single-step through a program, and to examine any of the 8085's internal registers. The monitor drives either the on-board keyboard/LED display or a serial TTY interface. This monitor was modified slightly in order to use the 8251A with a 2400 baud CRT as opposed to the 110 baud normally used. The 8273 program implements monitor-like user interface. 8273 commands are entered by a two-character code followed by any parameters required by that command. When 8273 interrupts occur, the source of the interrupt is displayed along with any results associated with it. To gain a flavor of how the user/program interface operates, a sample output is shown in Figure 43. The 8273 program prompt character is a "–" and user inputs are underlined.

The "SO 05" implements the Set Operating Mode command with a parameter of 05H. This sets the Buffer and Flag Stream modes. "SS 01" sets the 8273 in NRZI mode using the Set Serial I/O Mode command. The next command specifies General Receiver with a receiver buffer size of 0100H bytes (B_0 = 00, B_1 = 01). The "TF" command causes the 8273 to transmit a frame containing an address field of C2H and control field of 11H. The information field is 001122. The "TF" command has a special format. The L_0 and L_1 parameters are computed from the number of information field bytes entered.

After the TF command is entered, the 8273 transmits the frame (assuming that the modem protocol is observed). After the closing flag, the 8273 interrupts the 8085. The 8085 reads the interrupt results and places them in a buffer. The software examines this buffer for new results and if new results exist, the source of the interrupt is displayed along with the results.

In this example, the 0DH result indicates a Frame Complete interrupt. There is only one result for a transmitter interrupt, the interrupt's trailing zero results were included to simplify programming.

The next event is a frame reception. The interrupt results are displayed in the order read from the 8273. The E0H indicates a General Receive interrupt with the last byte of the information field received on an 8-bit boundary. The 03 00 (R_0, R_1) results show that there are 3H bytes of information field received. The remaining two results indicate that the received frame had a C2H address field and a 34H control field. The 3 bytes of information field are displayed on the next line.

```
8273 MONITOR V1.2

— SO 05
— SS 01
— GR 00 01
— TF C2 11 00 11 22
—
TxINT — 0D 00 00 00 00
—
RxINT — E0 03 00 C2 34
FF EE DD
—
                                611001-63
```

Figure 43. Sample 8273 Monitor I/O

Figures 44 through 51 show the flowcharts used for the 8273 program development. The actual program listing is included as Appendix A. Figure 44 is the main status poll loop. After all devices are initialized and a prompt character displayed, a loop is entered at LOOPIT. This loop checks for a change of status in the result buffer or if a keyboard character has been received by the 8251 or if a poll frame has been received. If any of these conditions are met, the program branches to the appropriate routine. Otherwise, the loop is traversed again.

The result buffer is implemented as a 255-byte circular buffer with two pointers: CNADR and LDADR. CNADR is the console pointer. It points to the next result to be displayed. LDADR is the load pointer. It points to the next empty position in the buffer into which the interrupt handler places the next result. The same buffer is used for both transmitter and receiver results. LOOPIT examines these pointers to detect when CNADR is not equal to LDADR indicating that the buffer contains results which have not been displayed. When this occurs, the program branches to the DISPLY routine.

DISPLY determines the source of the undisplayed results by testing the first result. This first result is not necessarily the interrupt result code. If this result is 0CH or greater, the result is from a transmitter interrupt. Otherwise it is from a receiver source. The source of the result code is then displayed on the console along with the next four results from the buffer. If the source was a transmitter interrupt, the routine merely repoints the pointer CNADR and returns to LOOPIT. For a receiver source, the receiver data buffer is displayed in addition to the receiver interrupt results before returning to LOOPIT.

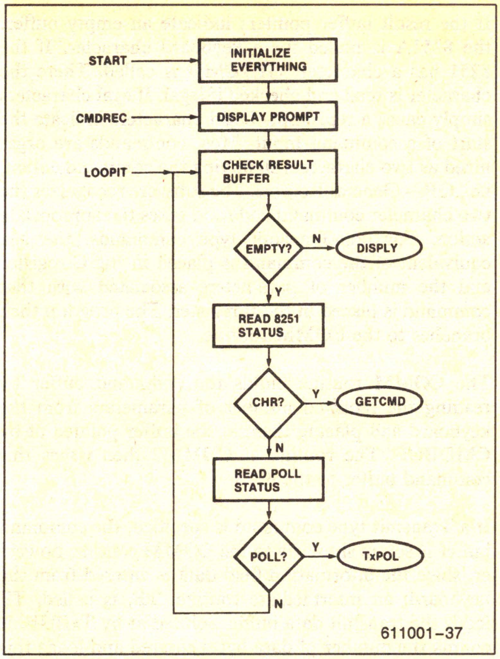


Figure 44. Main Status Poll Loop

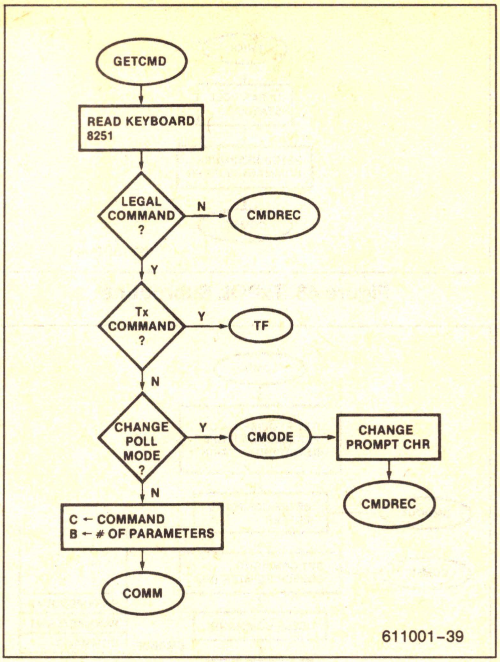


Figure 46. GETCMD Subroutine

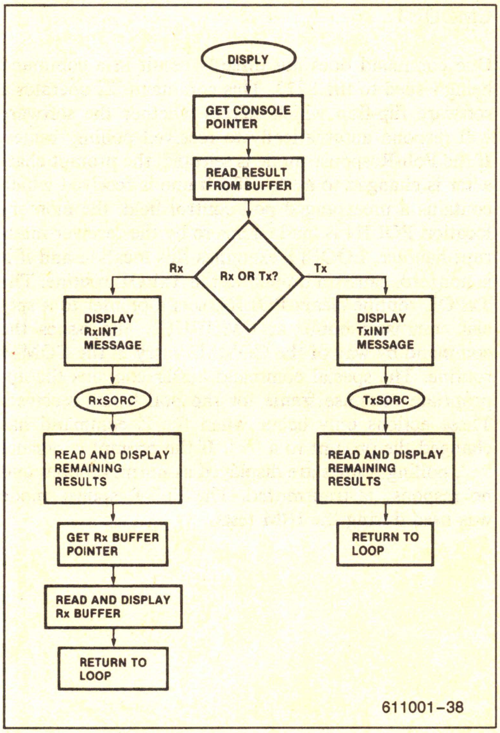


Figure 45. DISPLY Subroutine

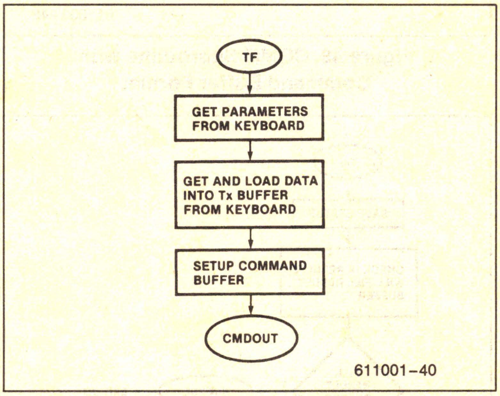


Figure 47. TF Subroutine

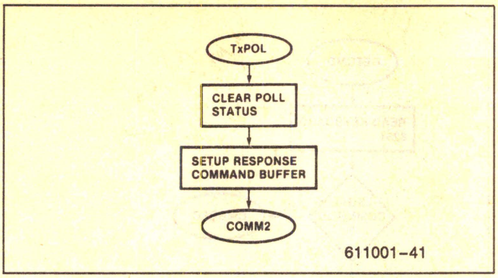


Figure 48. TxPOL Subroutine

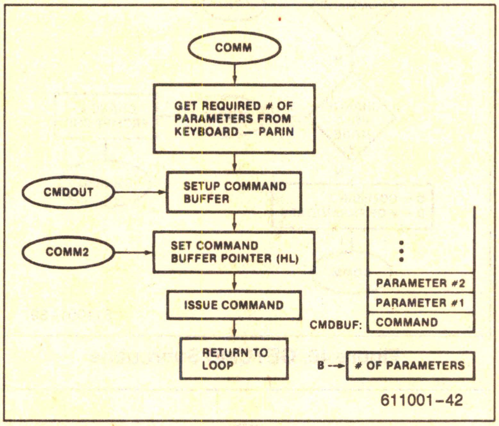


Figure 49. COMM Subroutine with Command Buffer Format

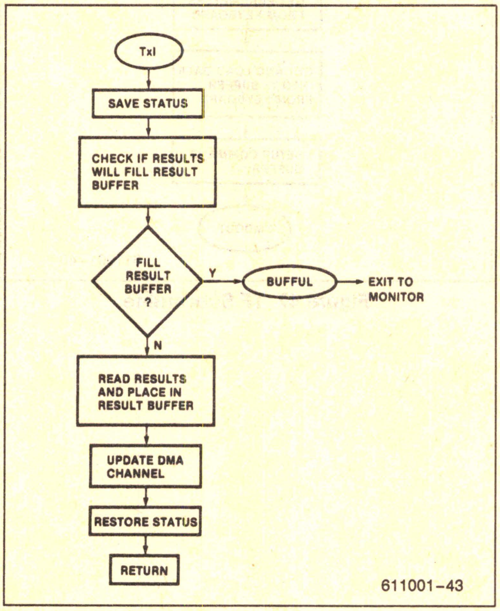


Figure 50. TxI (Transmitter Interrupt) Routine

If the result buffer pointers indicate an empty buffer, the 8251A is polled for a keyboard character. If the 8251 has a character, GETCMD is called. There the character is read and checked if legal. Illegal characters simply cause a reprompt. Legal characters indicate the start of a command input. Most commands are organized as two characters signifying the command action; i.e., GR—General Receive. The software recognizes the two character command code and takes the appropriate action. For non-Transmit type commands, the hex equivalent of the command is placed in the C register and the number of parameters associated with that command is placed in the B register. The program then branches to the COMM routine.

The COMM routine builds the command buffer by reading the required number of parameters from the keyboard and placing them at the buffer pointed at by CMDBUF. The routine at COMM2 then issues this command buffer to the 8273.

If a Transmit type command is specified, the command buffer is set up similarly to the COMM routine; however, since the information field data is entered from the keyboard, an intermediate routine, TF, is called. TF loads the transmit data buffer pointed at by TxBUF. It counts the number of data bytes entered and loads this number into the command buffer as L_0, L_1. The command is then issued to the 8273 by jumping to CMDOUT.

One command does not directly result in a command being issued to the 8273. This command, Z, operates a software flip-flop which selects whether the software will respond automatically to received polling frames. If the Poll-Response mode is selected, the prompt character is changed to a '+'. If a frame is received which contains a prearranged poll control field, the memory location POLIN is made nonzero by the receiver interrupt handler. LOOPIT examines this location and if it is nonzero, causes a branch to the TxPOL routine. The TxPOL routine clears POLIN, sets a pointer to a special command buffer at CMDBUF1, and issues the command by way of the COMM2 entry in the COMM routine. The special command buffer contains the appropriate response frame for the poll frame received. These actions only occur when the Z command has changed the prompt to a '+'. If the prompt is normal '−', polling frames are displayed as normal frames and no response is transmitted. The Poll-Response mode was used during the IBM tests.

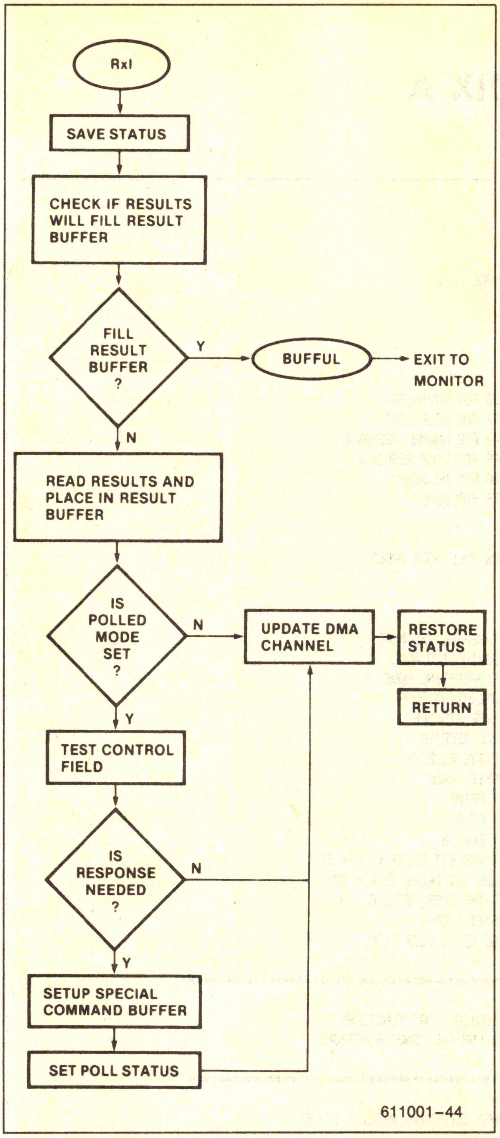


Figure 51. RxI (Receiver Interrupt) Routine

The final two software routines are the transmitter and receiver interrupt handlers. The transmit interrupt handler, TxI, simply saves the registers on the stack and checks if loading the result buffer will fill it. If the result buffer will overfill, the program is exited and control is passed to the SDK-85 monitor. If not, the results are read from the TxI/R register and placed in the result buffer at LDADR. The DMA pointers are then reset, the registers restored, and interrupts enabled. Execution then returns to the pre-interrupt location.

The receiver interrupt handler, RxI, is only slightly more complex. As in TxI, the registers are saved and the possibility of overfilling the result buffer is examined. If the result buffer is not full, the results are read from RxI/R and placed in the buffer. At this point the prompt character is examined to see if the Poll-Response mode is selected. If so, the control field is compared with two possible polling control fields. If there is a match, the special command buffer is loaded and the poll indicator, POLIN, is made nonzero. If no match occurred, no action is taken. Finally, the receiver DMA buffer pointers are reset, the processor status restored, and interrupts are enabled. The RET instruction returns execution to the pre-interrupt location.

This completes the discussion of the 8273/8085 system design.

CONCLUSION

This application note has covered the 8273 in some detail. The simple and low cost loop configuration was explored and an 8273/8085 system was presented as a sample design illustrating the DMA/interrupt-driven interface. It is hoped that the major features of the 8273, namely the frame-level command structure and the Digital Phase Locked Loop, have been shown to be a valuable asset in an SDLC system design.

APPENDIX A

```
ASM80 :F1:RAYT73.SRC


ISIS-II 8080/8085 MACRO ASSEMBLER, X108          MODULE   PAGE    1


  LOC  OBJ         SEQ         SOURCE STATEMENT

                     1 $NOPAGING MOD85  NOCOND
 0000                2 TRUE     EQU     00H                 ;00 FOR RAYTHEON
                     3 ;                                    ;FF FOR SELF-TEST
 0000                4 TRUE1    EQU     00H                 ;00 FOR NORMAL RESPONSE
                     5 ;                                    ;FF FOR LOOP RESPONSE
 0000                6 DEM      EQU     00H                 ;00 FOR NO DEMO
                     7 ;                                    ;FF FOR DEMO
                     8 ;
                     9 ;
                    10 ;GENERAL 8273 MONITOR WITH RAYTHEON POLL MODE ADDED
                    11 ;
                    17 ;
                    18 ;
                    19 ;COMMAND SUPPORTED ARE: RS - RESET SERIAL I/O MODE
                    20 ;                       SS - SET SERIAL I/O MODE
                    21 ;                       RO - RESET OPERATING MODE
                    22 ;                       SO - SET OPERATING MODE
                    23 ;                       RD - RECEIVER DISABLE
                    24 ;                       GR - GENERAL RECEIVE
                    25 ;                       SR - SELECTIVE RECEIVE
                    26 ;                       TF - TRANSMIT FRAME
                    27 ;                       AF - ABORT FRAME
                    28 ;                       SP - SET PORT B
                    29 ;                       RP - RESET PORT B
                    30 ;                       RB - RESET ONE BIT DELAY (PAR = 7F)
                    31 ;                       SB - SET ONE BIT DELAY (PAR = 80)
                    32 ;                       SL - SELECTIVE LOOP RECEIVE
                    33 ;                       TL - TRANSMIT LOOP
                    34 ;                       Z  - CHANGE MODES FLIP/FLOP
                    38 ;
                    39 ;*************************************************************************
                    40 ;
                    41 ;NOTE:  'SET' COMMANDS IMPLEMENT LOGICAL 'OR' FUNCTIONS
                    42 ;       'RESET' COMMANDS IMPLEMENT LOGICAL 'AND' FUNCTIONS
                    43 ;
                    44 ;*************************************************************************
                    45 ;
                    46 ;BUFFERED MODE MUST BE SELECTED WHEN SELECTIVE RECEIVE IS USED.
                    47 ;
                    48 ;COMMAND FORMAT IS: 'COMMAND (2 LTRS)' 'PAR.#1' 'PAR.#2' ETC.
                    49 ;
                    50 ;THE TRANSMIT FRAME COMMAND FORMAT IS: 'TF' 'A' 'C' 'BUFFER CONTENTS'.
                    51 ;       NO LENGTH COUNT IS NEEDED.  BUFFER CONTENTS IS ENDED WITH A CR.
                    52 ;
                    53 ;*************************************************************************
                    54 ;
                    55 ;POLLED MODE:   WHEN POLLED MODE IS SELECTED (DENOTED BY A '+' PROMPT), IF
```

611001-45

```
     ;              A SNRM-P OR RR(0)-P IS RECEIVED, A RESPONSE FRAME OF NSA-F
     ;              OR RR(0)-F IS TRANSMITTED.  OTHER COMMANDS OPERATE NORMALLY.
     ;
     ;**************************************************************************
     ;
     ;8273 EQUATES
     ;
0090 STAT73  EQU    90H              ;STATUS REGISTER
0090 COMM73  EQU    90H              ;COMMAND REGISTER
0091 PARM73  EQU    91H              ;PARAMETER REGISTER
0091 RESL73  EQU    91H              ;RESULT REGISTER
0092 TXIR73  EQU    92H              ;TX INTERRUPT RESULT REGISTER
0093 RXIR73  EQU    93H              ;RX INTERRUPT RESULT REGISTER
0092 TEST73  EQU    92H              ;TEST MODE REGISTER
0020 CPBF    EQU    20H              ;PARAMETER BUFFER FULL BIT
0004 TXINT   EQU    04H              ;TX INTERRUPT BIT IN STATUS REGISTER
0008 RXINT   EQU    08H              ;RX INTERRUPT BIT IN STATUS REGISTER
0001 TXIRA   EQU    01H              ;TX INT RESULT AVAILABLE BIT
0002 RXIRA   EQU    02H              ;RX INT RESULT AVAILABLE BIT
     ;
     ;8253 EQUATES
     ;
009B MODE53  EQU    9BH              ;8253 MODE WORD REGISTER
009C CNT053  EQU    9CH              ;COUNTER 0 REGISTER
009D CNT153  EQU    9DH              ;COUNTER 1 REGISTER
009E CNT253  EQU    9EH              ;COUNTER 2 REGISTER
000C COBR    EQU    000CH            ;CONSOLE BAUD RATE (2400)
0036 MDCNT0  EQU    36H              ;MODE FOR COUNTER 0
00B6 MDCNT2  EQU    0B6H             ;MODE FOR COUNTER 2
2017 LKBR1   EQU    2017H            ;8273 BAUD RATE LSB ADR
2018 LKBR2   EQU    2018H            ;8273 BAUD RATE MSB ADR
     ;
     ;BAUD RATE TABLE:    BAUD RATE      LKBR1   LKBR2
     ;                    *********      *****   *****
     ;                       9600         2E      00
     ;                       4800         5C      00
     ;                       2400         B9      00
     ;                       1200         72      01
     ;                        600         E5      02
     ;                        300         C9      05
     ;
     ;
     ;8257 EQUATES
     ;
00A8 MODE57  EQU    0A8H             ;8257 MODE PORT
00A0 CH0ADR  EQU    0A0H             ;CH0 (RX) ADR REGISTER
00A1 CH0TC   EQU    0A1H             ;CH0 TERMINAL COUNT REGISTER
00A2 CH1ADR  EQU    0A2H             ;CH1 (TX) ADR REGISTER
00A3 CH1TC   EQU    0A3H             ;CH1 TERMINAL COUNT REGISTER
00A8 STAT57  EQU    0A8H             ;STATUS REGISTER
8200 RXBUF   EQU    8200H            ;RX BUFFER START ADDRESS
8000 TXBUF   EQU    8000H            ;TX BUFFER START ADDRESS
0062 DRDMA   EQU    62H              ;DISABLE RX DMA CHANNEL, TX STILL ON
41FF RXTC    EQU    41FFH            ;TERMINAL COUNT AND MODE FOR RX CHANNEL
0063 ENDMA   EQU    63H              ;ENABLE BOTH TX AND RX CHANNELS-EXT. WR, TX STOP
0061 DTDMA   EQU    61H              ;DISABLE TX DMA CHANNEL, RX STILL ON
81FF TXTC    EQU    81FFH            ;TERMINAL COUNT AND MODE FOR TX CHANNEL
     ;
```

611001-46

```
                 ;8251A EQUATES
                 ;
0089             CNTL51  EQU     89H             ;CONTROL WORD REGISTER
0089             STAT51  EQU     89H             ;STATUS REGISTER
0088             TXD51   EQU     88H             ;TX DATA REGISTER
0088             RXD51   EQU     88H             ;RX DATA REGISTER
00CE             MDE51   EQU     0CEH            ;MODE 16X,2 STOP,NO PARITY
0027             CMD51   EQU     27H             ;COMMAND, ENABLE TX&RX
0002             RDY     EQU     02H             ;RXRDY BIT
                 ;
                 ;MONITOR SUBROUTINE EQUATES
                 ;
061F             GETCH   EQU     061FH           ;GET CHR FROM KEYBOARD, ASCII IN CH
05F8             ECHO    EQU     05F8H           ;ECHO CHR TO DISPLAY
075E             VALDG   EQU     075EH           ;CHECK IF VALID DIGIT, CARRY SET IF VALID
05BB             CNVBN   EQU     05BBH           ;CONVERTS ASCII TO HEX
05EB             CRLF    EQU     05EBH           ;DISPLAY CR, HENCE LF TOO
06C7             NMOUT   EQU     06C7H           ;CONVERT BYTE TO 2 ASCII CHR AND DISPLAY
                 ;
                 ;MISC EQUATES
                 ;
20C0             STKSRT  EQU     20C0H           ;STACK START
0003             CNTLC   EQU     03H             ;CNTL-C EQUIVALENT
0008             MONTOR  EQU     0008H           ;MONITOR
2000             CMDBUF  EQU     2000H           ;START OF COMMAND BUFFER
2020             CMDBF1  EQU     2020H           ;POLL MODE SPECIAL TX COMMAND BUFFER
000D             CR      EQU     0DH             ;ASCII CR
000A             LF      EQU     0AH             ;ASCII LF
20D4             RST75   EQU     20D4H           ;RST7.5 JUMP ADDRESS
20CE             RST65   EQU     20CEH           ;RST6.5 JUMP ADDRESS
2010             LDADR   EQU     2010H           ;RESULT BUFFER LOAD POINTER STORAGE
2013             CNADR   EQU     2013H           ;RESULT BUFFER CONSOLE POINTER STORAGE
2800             RESBUF  EQU     2800H           ;RESULT BUFFER START - 255 BYTES
0093             SNRMP   EQU     93H             ;SNRM-P CONTROL CODE
0011             RR0P    EQU     11H             ;RR(0)-P CONTROL CODE
0073             NSAF    EQU     73H             ;NSA-F CONTROL CODE
0011             RR0F    EQU     11H             ;RR(0)-F CONTROL CODE
2015             PRMPT   EQU     2015H           ;PRMPT STORAGE
2016             POLIN   EQU     2016H           ;POLL MODE SELECTION INDICATOR
2027             DEMODE  EQU     2027H           ;DEMO MODE INDICATOR
                 ;
                 ;*************************************************************************
                 ;
                 ;RAM STORAGE DEFINITIONS:
                 ;       LOC             DEF
                 ;       ---             ---
                 ;       2000-200F       COMMAND BUFFER
                 ;       2010-2011       RESULT BUFFER LOAD POINTER
                 ;       2013-2014       RESULT BUFFER CONSOLE POINTER
                 ;       2015            PROMPT CHARACTER STORAGE
                 ;       2016            POLL MODE INDICATOR
                 ;       2017            BAUD RATE LSB FOR SELF-TEST
                 ;       2018            BAUD RATE MSB FOR SELF-TEST
                 ;       2019            SPARE
                 ;       2020-2026       RESPONSE COMMAND BUFFER FOR POLL MODE
                 ;       2800-28FF       RESULT BUFFER
                 ;
                 ;*************************************************************************
```

611001-47

```
                 ;
                 ;PROGRAM START
                 ;
                 ;INITIALIZE 8253, 8257, 8251A, AND RESET 8273.
                 ;ALSO SET NORMAL MODE, AND PRINT SIGNON MESSAGE
                 ;
0800                     ORG     800H

0800 31C020      START:  LXI     SP,STKSRT       ;INITIALIZE SP
0803 3E36                MVI     A,MDCNT0        ;8253 MODE SET
0805 D39B                OUT     MODE53          ;8253 MODE PORT
0807 3A1720              LDA     LKBR1           ;GET 8273 BAUD RATE LSB
080A D39C                OUT     CNT053          ;USING COUNTER 0 AS BAUD RATE GEN
080C 3A1820              LDA     LKBR2           ;GET 8273 BUAD RATE MSB
080F D39C                OUT     CNT053          ;COUNTER 0
0811 CD1A0B              CALL    RXDMA           ;INITIALIZE 8257 RX DMA CHANNEL
0814 CD350B              CALL    TXDMA           ;INITIALIZE 8257 TX DMA CHANNEL
0817 3E01                MVI     A,01H           ;OUTPUT 1 FOLLOWED BY A 0
0819 D392                OUT     TEST73          ;TO TEST MODE REGISTER
081B 3E00                MVI     A,00H           ;TO RESET THE 8273
081D D392                OUT     TEST73
081F 3E2D                MVI     A,'-'           ;NORMAL MODE PROMPT CHR
0821 321520              STA     PRMPT           ;PUT IN STORAGE
0824 3E00                MVI     A,00H           ;TX POLL RESPONSE INDICATOR
0826 321620              STA     POLIN           ;0 MEANS NO SPECIAL TX
0829 322720              STA     DEMODE          ;CLEAR DEMO MODE
082C 21A30C              LXI     H,SIGNON                ;SIGNON MESSAGE ADR
082F CD920C              CALL    TYMSG           ;DISPLAY SIGNON
                 ;
                 ;MONITOR USES JUMPS IN RAM TO DIRECT INTERRUPTS
                 ;
0832 21D420              LXI     H,RST75         ;RST7.5 JUMP LOCATION USED BY MONITOR
0835 01000C              LXI     B,RXI           ;ADDRESS OF RX INT ROUTINE
0838 36C3                MVI     M,0C3H          ;LOAD 'JMP' OPCODE
083A 23                  INX     H               ;INC POINTER
083B 71                  MOV     M,C             ;LOAD RXI LSB
083C 23                  INX     H               ;INC POINTER
083D 70                  MOV     M,B             ;LOAD RXI MSB
083E 21CE20              LXI     H,RST65         ;RST6.5 JUMP LOCATION USED BY MONITOR
0841 01CE0C              LXI     B,TXI           ;ADDRESS OF TX INT ROUTINE
0844 36C3                MVI     M,0C3H          ;LOAD 'JMP' OPCODE
0846 23                  INX     H               ;INC POINTER
0847 71                  MOV     M,C             ;LOAD TXI LSB
0848 23                  INX     H               ;INC POINTER
0849 70                  MOV     M,B             ;LOAD TXI MSB
084A 3E18                MVI     A,18H           ;GET SET TO RESET INTERRUPTS
084C 30                  SIM                     ;RESET INTERRUPTS
084D FB                  EI                      ;ENABLE INTERRUPTS
                 ;
                 ;INITIALIZE BUFFER POINTER
                 ;
                 ;
084E 210028              LXI     H,RESBUF        ;SET RESULT BUFFER POINTERS
0851 221320              SHLD    CNADR           ;RESULT CONSOLE POINTER
0854 221020              SHLD    LDADR           ;RESULT LOAD POINTER
                 ;
                 ;MAIN PROGRAM LOOP - CHECKS FOR CHANGE IN RESULT POINTERS, USART STATUS,
                 ;       OR POLL STATUS
```

611001-48

```
                    ;
0857 CDEB05     CMDREC: CALL  CRLF        ;DISPLAY CR
085A 3A1520             LDA   PRMPT       ;GET CURRENT PROMPT CHR
085D 4F                 MOV   C,A         ;MOVE TO C
085E CDF805             CALL  ECHO        ;DISPLAY IT
0861 2A1320     LOOPIT: LHLD  CNADR       ;GET CONSOLE POINTER
0864 7D                 MOV   A,L         ;SAVE POINTER LSB
0865 2A1020             LHLD  LDADR       ;GET LOAD POINTER
0868 BD                 CMP   L           ;SAME LSB?
0869 C2390A             JNZ   DISPY       ;NO, RESULTS NEED DISPLAYING
086C DB89               IN    STAT51      ;YES, CHECK KEYBOARD
086E E602               ANI   RDY         ;CHR RECEIVED?
0870 C27D08             JNZ   GETCMD      ;MUST BE CHR SO GO GET IT
0873 3A1620             LDA   POLIN       ;GET POLL MODE STATUS
0876 A7                 ANA   A           ;IS IT 0?
0877 C24C09             JNZ   TXPOL       ;NO, THEN POLL OCCURRED
087A C36108             JMP   LOOPIT      ;YES, TRY AGAIN
                    ;
                    ;
                    ;COMMAND RECOGNIZER ROUTINE
                    ;
                    ;
087D CD1F06     GETCMD: CALL  GETCH       ;GET CHR
0880 CDF805             CALL  ECHO        ;ECHO IT
0883 79                 MOV   A,C         ;SETUP FOR COMPARE
0884 FE52               CPI   'R'         ;R?
0886 CAAF08             JZ    RDWN        ;GET MORE
0889 FE53               CPI   'S'         ;S?
088B CAD708             JZ    SDWN        ;GET MORE
088E FE47               CPI   'G'         ;G?
0890 CAFF08             JZ    GDWN        ;GET MORE
0893 FE54               CPI   'T'         ;T?
0895 CA0E09             JZ    TDWN        ;GET MORE
0898 FE41               CPI   'A'         ;A?
089A CA2209             JZ    ADWN        ;GET MORE
089D FE5A               CPI   'Z'         ;Z?
089F CA3109             JZ    CMODE       ;YES, GO CHANGE MODE
08A2 FE03               CPI   CNTLC       ;CNTL-C?
08A4 CA0800             JZ    MONTOR      ;EXIT TO MONITOR
08A7 0E3F       ILLEG:  MVI   C,'?'       ;PRINT ?
08A9 CDF805             CALL  ECHO        ;DISPLAY IT
08AC C35708             JMP   CMDREC      ;LOOP FOR COMMAND

08AF CD1F06     RDWN:   CALL  GETCH       ;GET NEXT CHR
08B2 CDF805             CALL  ECHO        ;ECHO IT
08B5 79                 MOV   A,C         ;SETUP FOR COMPARE
08B6 FE4F               CPI   'O'         ;O?
08B8 CA5D09             JZ    ROCMD       ;RO COMMAND
08BB FE53               CPI   'S'         ;S?
08BD CA6709             JZ    RSCMD       ;RS COMMAND
08C0 FE44               CPI   'D'         ;D?
08C2 CA7109             JZ    RDCMD       ;RD COMMAND
08C5 FE50               CPI   'P'         ;P?
08C7 CAD809             JZ    RPCMD       ;RP COMMAND
08CA FE52               CPI   'R'         ;R?
08CC CA0008             JZ    START       ;START OVER
08CF FE42               CPI   'B'         ;B?
08D1 CA7B09             JZ    RBCMD       ;RB COMMAND
```

611001-49

```
08D4 C3A708              JMP    ILLEG          ;ILLEGAL, TRY AGAIN

08D7 CD1F06      SDWN:   CALL   GETCH          ;GET NEXT CHR
08DA CDF805              CALL   ECHO           ;ECHO IT
08DD 78                  MOV    A,B            ;SETUP FOR COMPARE
08DE FE4F                CPI    'O'            ;O?
08E0 CAA609              JZ     SOCMD          ;SO COMMAND
08E3 FE53                CPI    'S'            ;S?
08E5 CAB009              JZ     SSCMD          ;SS COMMAND
08E8 FE52                CPI    'R'            ;R?
08EA CABA09              JZ     SRCMD          ;SR COMMAND
08ED FE50                CPI    'P'            ;P?
08EF CAE209              JZ     SPCMD          ;SP COMMAND
08F2 FE42                CPI    'B'            ;B?
08F4 CA8509              JZ     SBCMD          ;SB COMMAND
08F7 FE4C                CPI    'L'            ;L?
08F9 CA8F09              JZ     SLCMD          ;SL COMMAND
08FC C3A708              JMP    ILLEG          ;ILLEGAL, TRY AGAIN

08FF CD1F06      GDWN:   CALL   GETCH          ;GET NEXT CHR
0902 CDF805              CALL   ECHO           ;ECHO IT
0905 78                  MOV    A,B            ;SETUP FOR COMPARE
0906 FE52                CPI    'R'            ;R?
0908 CAC409              JZ     GRCMD          ;GR COMMAND
090B C3A708              JMP    ILLEG          ;ILLEGAL, TRY AGAIN

090E CD1F06      TDWN:   CALL   GETCH          ;GET NEXT CHR
0911 CDF805              CALL   ECHO           ;ECHO IT
0914 78                  MOV    A,B            ;SETUP FOR COMPARE
0915 FE46                CPI    'F'            ;F?
0917 CAEC09              JZ     TFCMD          ;TF COMMAND
091A FE4C                CPI    'L'            ;L?
091C CA9909              JZ     TLCMD          ;TL COMMAND
091F C3A708              JMP    ILLEG          ;ILLEGAL, TRY AGAIN

0922 CD1F06      ADWN:   CALL   GETCH          ;GET NEXT CHR
0925 CDF805              CALL   ECHO           ;ECHO IT
0928 78                  MOV    A,B            ;SETUP FOR COMPARE
0929 FE46                CPI    'F'            ;F?
092B CACE09              JZ     AFCMD          ;AF COMMAND
092E C3A708              JMP    ILLEG          ;ILLEGAL, TRY AGAIN
                 ;
                 ;RESET POLL MODE RESPONSE - CHANGE PROMPT CHR AS INDICATOR
                 ;
0931 F3          CMODE:  DI                    ;DISABLE INTERRUPTS
0932 3A1520              LDA    PRMPT          ;GET CURRENT PROMPT
0935 FE2D                CPI    '-'            ;NORMAL MODE?
0937 C24309              JNZ    SW             ;NO, CHANGE IT
093A 3E2B                MVI    A,'+'          ;NEW PROMPT
093C 321520              STA    PRMPT          ;STORE NEW PROMPT
093F FB                  EI                    ;ENABLE INTERRUPTS
0940 C35708              JMP    CMDREC         ;RETURN TO LOOP
0943 3E2D        SW:     MVI    A,'-'          ;NEW PROMPT CHR
0945 321520              STA    PRMPT          ;STORE IT
0948 FB                  EI                    ;ENABLE INTERRUPTS
0949 C35708              JMP    CMDREC         ;RETURN TO LOOP
                 ;
                 ;
```

611001-50

```
                 ;TRANSMIT ANSWER TO POLL SETUP
                 ;
094C 3E00        TXPOL:  MVI     A,00H          ;CLEAR POLL INDICATOR
094E 321620              STA     POLIN          ;INDICATOR ADR
0951 216108              LXI     H,LOOPIT       ;SETUP STACK FOR COMMAND OUTPUT
0954 E5                  PUSH    H              ;PUT RETURN TO CMDREC ON STACK
0955 0604                MVI     B,04H          ;GET # OF PARAMETERS READY
0957 212020              LXI     H,CMDBF1       ;POINT TO SPECIAL BUFFER
095A C3FF0A              JMP     COMM2          ;JUMP TO COMMAND OUTPUTER
                 ;
                 ;
                 ;
                 ;COMMAND IMPLEMENTING ROUTINES
                 ;
                 ;
                 ;RO - RESET OPERATING MODE
                 ;
095D 0601        ROCMD:  MVI     B,01H          ;# OF PARAMETERS
095F 0E51                MVI     C,51H          ;COMMAND
0961 CDE50A              CALL    COMM           ;GET PARAMETERS AND ISSUE COMMAND
0964 C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;RS - RESET SERIAL I/O MODE COMMAND
                 ;
0967 0601        RSCMD:  MVI     B,01H          ;# OF PARAMETERS
0969 0E60                MVI     C,60H          ;COMMAND
096B CDE50A              CALL    COMM           ;GET PARAMETERS AND ISSUE COMMAND
096E C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;RD - RECEIVER DISABLE COMMAND
                 ;
0971 0600        RDCMD:  MVI     B,00H          ;# OF PARAMETERS
0973 0EC5                MVI     C,0C5H         ;COMMAND
0975 CDE50A              CALL    COMM           ;ISSUE COMMAND
0978 C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;RB - RESET ONE BIT DELAY COMMAND
                 ;
097B 0601        RBCMD:  MVI     B,01H          ;# OF PARAMETERS
097D 0E64                MVI     C,64H          ;COMMAND
097F CDE50A              CALL    COMM           ;GET PARAMETER AND ISSUE COMMAND
0982 C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;SB - SET ONE BIT DELAY COMMAND
                 ;
0985 0601        SBCMD:  MVI     B,01H          ;# OF PARAMETERS
0987 0EA4                MVI     C,0A4H         ;COMMAND
0989 CDE50A              CALL    COMM           ;GET PARAMETER AND ISSUE COMMAND
098C C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;SL - SELECTIVE LOOP RECEIVE COMMAND
                 ;
098F 0604        SLCMD:  MVI     B,04H          ;# OF PARAMETERES
0991 0EC2                MVI     C,0C2H         ;COMMAND
0993 CDE50A              CALL    COMM           ;GET PARAMETERS AND ISSUE COMMAND
0996 C35708              JMP     CMDREC         ;GET NEXT COMMAND
                 ;
                 ;TL - TRANSMIT LOOP COMMAND
```

611001-51

```
                  ;
0999 210020       TLCMD:  LXI     H,CMDBUF        ;SET COMMAND BUFFER POINTER
099C 0602                 MVI     B,02H           ;LOAD PARAMETER COUNTER
099E 36CA                 MVI     M,0CAH          ;LOAD COMMAND INTO BUFFER
09A0 210220               LXI     H,CMDBUF+2      ;POINT AT ADR AND CNTL POSITIONS
09A3 C3F609               JMP     TFCMD1          ;FINISH OFF COMMAND IN TF ROUTINE
                  ;
                  ;SO - SET OPERATING MODE COMMAND
                  ;
09A6 0601         SOCMD:  MVI     B,01H           ;# OF PARAMETERS
09A8 0E91                 MVI     C,91H           ;COMMAND
09AA CDE50A               CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09AD C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;SS - SET SERIAL I/O COMMAND
                  ;
09B0 0601         SSCMD:  MVI     B,01H           ;# OF PARAMETERS
09B2 0EA0                 MVI     C,0A0H          ;COMMAND
09B4 CDE50A               CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09B7 C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;SR - SELECTIVE RECEIVE COMMAND
                  ;
09BA 0604         SRCMD:  MVI     B,04H           ;# OF PARAMETERS
09BC 0EC1                 MVI     C,0C1H          ;COMMAND
09BE CDE50A               CALL    COMM            ;GET PARAMETERS AND ISSUE COMMAND
09C1 C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;GR - GENERAL RECEIVE COMMAND
                  ;
09C4 0602         GRCMD:  MVI     B,02H           ;NO PARAMETERS
09C6 0EC0                 MVI     C,0C0H          ;COMMAND
09C8 CDE50A               CALL    COMM            ;ISSUE COMMAND
09CB C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;AF - ABORT FRAME COMMAND
                  ;
09CE 0600         AFCMD:  MVI     B,00H           ;NO PARAMETERS
09D0 0ECC                 MVI     C,0CCH          ;COMMAND
09D2 CDE50A               CALL    COMM            ;ISSUE COMMAND
09D5 C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;RP - RESET PORT COMMAND
                  ;
09D8 0601         RPCMD:  MVI     B,01H           ;# OF PARAMETERS
09DA 0E63                 MVI     C,63H           ;COMMAND
09DC CDE50A               CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09DF C35708               JMP     CMDREC          ;GET NEXT COMMAND
                  ;
                  ;SP - SET PORT COMMAND
                  ;
09E2 0601         SPCMD:  MVI     B,01H           ;# OF PARAMETERS
09E4 0EA3                 MVI     C,0A3H          ;COMMAND
09E6 CDE50A               CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09E9 C35708               JMP     CMDREC          ;GET NEX COMMAND
                  ;
                  ;TF - TRANSMIT FRAME COMMAND
                  ;
```

611001-52

```
09EC 210020      TFCMD:  LXI     H,CMDBUF        ;SET COMMAND BUFFER POINTER
09EF 0602                MVI     B,02H           ;LOAD PARAMETER COUNTER
09F1 36C8                MVI     M,0C8H          ;LOAD COMMAND INTO BUFFER
09F3 210220              LXI     H,CMDBUF+2      ;POINT AT ADR AND CNTL POSITIONS
09F6 78          TFCMD1: MOV     A,B             ;TEST PARAMETER COUNT
09F7 A7                  ANA     A               ;IS IT 0?
09F8 CA070A              JZ      TBUFL           ;YES, LOAD TX DATA BUFFER
09FB CDAD0A              CALL    PARIN           ;GET PARAMETER
09FE DAA708              JC      ILLEG           ;ILLEGAL CHR RETURNED
0A01 23                  INX     H               ;INC COMMAND BUFFER POINTER
0A02 05                  DCR     B               ;DEC PARAMETER COUNTER
0A03 77                  MOV     M,A             ;LOAD PARAMETER INTO COMMAND BUFFER
0A04 C3F609              JMP     TFCMD1          ;GET NEXT PARAMETER

0A07 210080      TBUFL:  LXI     H,TXBUF         ;LOAD TX DATA BUFFER POINTER
0A0A 010000              LXI     B,0000H         ;CLEAR BC - BYTE COUNTER
0A0D C5          TBUFL1: PUSH    B               ;SAVE BYTE COUNTER
0A0E CDAD0A              CALL    PARIN           ;GET DATA, ALIAS PARAMETER
0A11 DA1B0A              JC      ENDCHK          ;MAYBE END IF ILLEGAL
0A14 77                  MOV     M,A             ;LOAD DATA BYTE INTO BUFFER
0A15 23                  INX     H               ;INC BUFFER POINTER
0A16 C1                  POP     B               ;RESTORE BYTE COUNTER
0A17 03                  INX     B               ;INC BYTE COUNTER
0A18 C30D0A              JMP     TBUFL1          ;GET NEXT DATA
0A1B FE0D        ENDCHK: CPI     CR              ;RETURNED ILLEGAL CHR CR?
0A1D CA240A              JZ      TBUFFL          ;YES, THEN TX BUFFER FULL
0A20 C1                  POP     B               ;RESTORE B TO SAVE STACK
0A21 C3A708              JMP     ILLEG           ;ILLEGAL CHR
0A24 C1          TBUFFL: POP     B               ;RESTORE BYTE COUNTER
0A25 210120              LXI     H,CMDBUF+1      ;POINT INTO COMMAND BUFFER
0A28 71                  MOV     M,C             ;STORE BYTE COUNT LSB
0A29 23                  INX     H               ;INC POINTER
0A2A 70                  MOV     M,B             ;STORE BYTE COUNT MSB
0A2B 0604                MVI     B,04H           ;LOAD PARAMETER COUNT INTO B
0A2D 21360A              LXI     H,TFRET         ;GET RETURN ADR FOR THIS ROUTINE
0A30 C5                  PUSH    B               ;PUSH ONCE
0A31 E3                  XTHL                    ;PUT RETURN ON STACK
0A32 C5                  PUSH    B               ;PUSH IT SO CMDOUT CAN USE IT
0A33 C3FB0A              JMP     CMDOUT          ;ISSUE COMMAND
0A36 C35708      TFRET:  JMP     CMDREC          ;GET NEXT COMMAND
                 ;
                 ;
                 ;ROUTINE TO DISPLAY RESULT IN RESULT BUFFER WHEN LOAD AND CONSOLE
                 ;POINTERS ARE DIFFERENT.
                 ;
                 ;
0A39 1605        DISPY:  MVI     D,05H           ;D IS RESULT COUNTER
0A3B 2A1320              LHLD    CNADR           ;GET CONSOLE POINTER
0A3E E5                  PUSH    H               ;SAVE IT
0A3F 7E                  MOV     A,M             ;GET RESULT IC
0A40 E61F                ANI     1FH             ;LIMIT TO RESULT CODE
0A42 FE0C                CPI     0CH             ;TEST IF RX OR TX SOURCE
0A44 DA620A              JC      RXSORC          ;CARRY, THEN RX SOURCE
0A47 21C30C      TXSORC: LXI     H,TXIMSG        ;TX INT MESSAGE
0A4A CD920C              CALL    TYMSG           ;DISPLAY IT
0A4D E1          DISPY2: POP     H               ;RESTORE CONSOLE POINTER
0A4E 7E          DISPY1: MOV     A,M             ;GET RESULT
0A4F CDC706              CALL    NMOUT           ;CONVERT AND DISPLAY
```

611001-53

```
0A52 0E20               MVI     C,' '           ;SP CHR
0A54 CDF805             CALL    ECHO            ;DISPLAY IT
0A57 2C                 INR     L               ;INC BUFFER POINTER
0A58 15                 DCR     D               ;DEC RESULT COUNTER
0A59 C24E0A             JNZ     DISPY1          ;NOT DONE
0A5C 221320             SHLD    CNADR           ;UPDATE CONSOLE POINTER
0A5F C35708             JMP     CMDREC          ;RETURN TO LOOP
                ;
                ;
                ;RECEIVER SOURCE - DISPLAY RESULTS AND RECEVIE BUFFER CONTENTS
                ;
                ;
0A62 21B80C     RXSORC: LXI     H,RXIMSG        ;RX INT MESSAGE ADR
0A65 CD920C             CALL    TYMSG           ;DISPLAY MESSAGE
0A68 E1                 POP     H               ;RESTORE CONSOLE POINTER
0A69 7E         RXS1:   MOV     A,M             ;RETRIEVE RESULT FROM BUFFER
0A6A CDC706             CALL    NMOUT           ;CONVERT AND DISPLAY IT
0A6D 0E20               MVI     C,' '           ;ASCII SP
0A6F CDF805             CALL    ECHO            ;DISPLAY IT
0A72 2C                 INR     L               ;INC CONSOLE POINTER
0A73 15                 DCR     D               ;DEC RESULT COUNTER
0A74 7A                 MOV     A,D             ;GET SET TO TEST COUNTER
0A75 FE04               CPI     04H             ;IS THE RESULT R0?
0A77 CAA20A             JZ      R0PT            ;YES, GO SAVE IT
0A7A FE03               CPI     03H             ;IS THE RESULT R1?
0A7C CAA70A             JZ      R1PT            ;YES, GO SAVE IT
0A7F A7         RXS2:   ANA     A               ;TEST RESULT COUNTER
0A80 C2690A             JNZ     RXS1            ;NOT DONE YET, GET NEXT RESULT
0A83 221320             SHLD    CNADR           ;DONE, SO UPDATE CONSOLE POINTER
0A86 CDEB05             CALL    CRLF            ;DISPLAY CR
0A89 210082             LXI     H,RXBUF         ;POINT AT RX BUFFER
0A8C C1                 POP     B               ;RETRIEVE RECEIVED COUNT
0A8D 78         RXS3:   MOV     A,B             ;IS COUNT 0?
0A8E B1                 ORA     C               ;
0A8F CA5708             JZ      CMDREC          ;YES, GO BACK TO LOOP
0A92 7E                 MOV     A,M             ;NO, GET CHR
0A93 C5                 PUSH    B               ;SAVE BC
0A94 CDC706             CALL    NMOUT           ;CONVERT AND DISPLAY CHR
0A97 0E20               MVI     C,' '           ;ASCII SP
0A99 CDF805             CALL    ECHO            ;DISPLAY IT TO SEPARATE DATA
0A9C C1                 POP     B               ;RESTORE BC
0A9D 0B                 DCX     B               ;DEC COUNT
0A9E 23                 INX     H               ;INC POINTER
0A9F C38D0A             JMP     RXS3            ;GET NEXT CHR

0AA2 4E         R0PT:   MOV     C,M             ;GET R0 FOR RESULT BUFFER
0AA3 C5                 PUSH    B               ;SAVE IT
0AA4 C37F0A             JMP     RXS2            ;RETURN

0AA7 C1         R1PT:   POP     B               ;GET R0
0AA8 46                 MOV     B,M             ;GET R1 FOR RESULT BUFFER
0AA9 C5                 PUSH    B               ;SAVE IT
0AAA C37F0A             JMP     RXS2
                ;
                ;
                ;
                ;PARAMETER INPUT - PARAMETER RETURNED IN E REGISTER
                ;
```

611001-54

```
                  ;
0AAD C5           PARIN:  PUSH   B          ;SAVE BC
0AAE 1601                 MVI    D,01H      ;SET CHR COUNTER
0AB0 CD1F06               CALL   GETCH      ;GET CHR
0AB3 CDF805               CALL   ECHO       ;ECHO IT
0AB6 79                   MOV    A,C        ;PUT CHR IN A
0AB7 FE20                 CPI    ' '        ;SP?
0AB9 C2E00A               JNZ    PARIN1     ;NO, ILLEGAL, TRY AGAIN
0ABC CD1F06       PARIN3: CALL   GETCH      ;GET CHR OF PARAMETER
0ABF CDF805               CALL   ECHO       ;ECHO IT
0AC2 CD5E07               CALL   VALDG      ;IS IT A VALID CHR?
0AC5 D2E00A               JNC    PARIN1     ;NO, TRY AGAIN
0AC8 CDBB05               CALL   CNVBN      ;CONVERT IT TO HEX
0ACB 4F                   MOV    C,A        ;SAVE IT IN C
0ACC 7A                   MOV    A,D        ;GET CHR COUNTER
0ACD A7                   ANA    A          ;IS IT 0?
0ACE CADC0A               JZ     PARIN2     ;YES,DONE WITH THIS PARAMETER
0AD1 15                   DCR    D          ;DEC CHR COUNTER
0AD2 AF                   XRA    A          ;CLEAR CARRY
0AD3 79                   MOV    A,C        ;RECOVER 1ST CHR
0AD4 17                   RAL               ;ROTATE LEFT 4 PLACES
0AD5 17                   RAL
0AD6 17                   RAL
0AD7 17                   RAL
0AD8 5F                   MOV    E,A        ;SAVE IT IN E
0AD9 C3BC0A               JMP    PARIN3     ;GET NEXT CHR
0ADC 79           PARIN2: MOV    A,C        ;2ND CHR IN A
0ADD B3                   ORA    E          ;COMBINE BOTH CHRS
0ADE C1                   POP    B          ;RESTORE BC
0ADF C9                   RET               ;RETURN TO CALLING PROGRAM
0AE0 79           PARIN1: MOV    A,C        ;PUT ILLEGAL CHR IN A
0AE1 37                   STC               ;SET CARRY AS ILLEGAL STATUS
0AE2 C1                   POP    B          ;RESTORE BC
0AE3 C9                   RET               ;RETURN TO CALLING PROGRAM
                  ;
                  ;
                  ;JUMP HERE IF BUFFER FULL
                  ;
0AE4 CF           BUFFUL: DB     0CFH       ;EXIT TO MONITOR
                  ;
                  ;
                  ;COMMAND DISPATCHER
                  ;
                  ;
0AE5 210020       COMM:   LXI    H,CMDBUF   ;SET POINTER
0AE8 C5                   PUSH   B          ;SAVE BC
0AE9 71                   MOV    M,C        ;LOAD COMMAND INTO BUFFER
0AEA 78           COMM1:  MOV    A,B        ;CHECK PARAMETER COUNTER
0AEB A7                   ANA    A          ;IS IT 0?
0AEC CAFB0A               JZ     CMDOUT     ;YES, GO ISSUE COMMAND
0AEF CDAD0A               CALL   PARIN      ;GET PARAMETER
0AF2 DAA708               JC     ILLEG      ;ILLEGAL CHR RETURNED
0AF5 23                   INX    H          ;INC BUFFER POINTER
0AF6 05                   DCR    B          ;DEC PARAMETER COUNTER
0AF7 77                   MOV    M,A        ;PARAMETER TO BUFFER
0AF8 C3EA0A               JMP    COMM1      ;GET NEXT PARAMETER
0AFB 210020       CMDOUT: LXI    H,CMDBUF   ;REPOINT POINTER
0AFE C1                   POP    B          ;RESTORE PARAMETER COUNT
```

611001-55

```
0AFF DB90        COMM2:  IN      STAT73          ;READ 8273 STATUS
0B01 07                  RLC                     ;ROTATE CBSY INTO CARRY
0B02 DAFF0A              JC      COMM2           ;WAIT FOR OK
0B05 7E                  MOV     A,M             ;OK, MOVE COMMAND INTO A
0B06 D390                OUT     COMM73          ;OUTPUT COMMAND
0B08 78          PAR1:   MOV     A,B             ;GET PARAMETER COUNT
0B09 A7                  ANA     A               ;IS IT 0?
0B0A C8                  RZ                      ;YES, DONE, RETURN
0B0B 23                  INX     H               ;INC COMMAND BUFFER POINTER
0B0C 05                  DCR     B               ;DEC PARAMETER COUNT
0B0D DB90        PAR2:   IN      STAT73          ;READ STATUS
0B0F E620                ANI     CPBF            ;IS CPBF BIT SET?
0B11 C20D0B              JNZ     PAR2            ;WAIT TIL ITS 0
0B14 7E                  MOV     A,M             ;OK, GET PARAMETER FROM BUFFER
0B15 D391                OUT     PARM73          ;OUTPUT PARAMETER
0B17 C3080B              JMP     PAR1            ;GET NEXT PARAMETER
                 ;
                 ;
                 ;INITIALIZE AND ENABLE RX DMA CHANNEL
                 ;
                 ;
0B1A 3E62        RXDMA:  MVI     A,DRDMA         ;DISABLE RX DMA CHANNEL
0B1C D3A8                OUT     MODE57          ;8257 MODE PORT
0B1E 010082              LXI     B,RXBUF         ;RX BUFFER START ADDRESS
0B21 79                  MOV     A,C             ;RX BUFFER LSB
0B22 D3A0                OUT     CH0ADR          ;CH0 ADR PORT
0B24 78                  MOV     A,B             ;RX BUFFER MSB
0B25 D3A0                OUT     CH0ADR          ;CH0 ADR PORT
0B27 01FF41              LXI     B,RXTC          ;RX CH TEERMINAL COUNT
0B2A 79                  MOV     A,C             ;RX TERMINAL COUNT LSB
0B2B D3A1                OUT     CH0TC           ;CH0 TC PORT
0B2D 78                  MOV     A,B             ;RX TERMINAL COUNT MSB
0B2E D3A1                OUT     CH0TC           ;CH0 TC PORT
0B30 3E63                MVI     A,ENDMA         ;ENABLE DMA WORD
0B32 D3A8                OUT     MODE57          ;8257 MODE PORT
0B34 C9                  RET                     ;RETURN
                 ;
                 ;
                 ;INITIALIZE AND ENABLE TX DMA CHANNEL
                 ;
                 ;
0B35 3E61        TXDMA:  MVI     A,DTDMA         ;DISABLE TX DMA CHANNEL
0B37 D3A8                OUT     MODE57          ;8257 MODE PORT
0B39 010080              LXI     B,TXBUF         ;TX BUFFER START ADDRESS
0B3C 79                  MOV     A,C             ;TX BUFFER LSB
0B3D D3A2                OUT     CH1ADR          ;CH1 ADR PORT
0B3F 78                  MOV     A,B             ;TX BUFFER MSB
0B40 D3A2                OUT     CH1ADR          ;CH1 ADR PORT
0B42 01FF81      TXDMA1: LXI     B,TXTC          ;TX CH TERMINAL COUNT
0B45 79                  MOV     A,C             ;TX TERMINAL COUNT LSB
0B46 D3A3                OUT     CH1TC           ;CH1 TC PORT
0B48 78                  MOV     A,B             ;TX TERMINAL COUNT MSB
0B49 D3A3                OUT     CH1TC           ;CH1 TC PORT
0B4B 3E63                MVI     A,ENDMA         ;ENABLE DMA WORD
0B4D D3A8                OUT     MODE57          ;8257 MODE PORT
0B4F C9                  RET                     ;RETURN
                 ;
                 ;
```

611001-56

```
                ;INERRUPT PROCESSING SECTION
                ;
0C00                    ORG     0C00H
                ;
                ;
                ;RECEIVER INTERRUPT - RST 7.5 (LOC 3CH)
                ;
0C00 E5         RXI:    PUSH    H               ;SAVE HL
0C01 F5                 PUSH    PSW             ;SAVE PSW
0C02 C5                 PUSH    B               ;SAVE BC
0C03 D5                 PUSH    D               ;SAVE DE
0C04 3E62               MVI     A,DRDMA         ;DISABLE RX DMA
0C06 D3A8               OUT     MODE57          ;8257 MODE PORT
0C08 3E18               MVI     A,18H           ;RESET RST7.5 F/F
0C0A 30                 SIM
0C0B 1604               MVI     D,04H           ;D IS RESULT COUNTER
0C0D 2A1020             LHLD    LDADR           ;GET LOAD POINTER
0C10 E5                 PUSH    H               ;SAVE IT
0C11 E5                 PUSH    H               ;SAVE IT AGAIN
0C12 45                 MOV     B,L             ;SAVE LSB
0C13 2A1320             LHLD    CNADR           ;GET CONSOLE POINTER
0C16 04         RXI1:   INR     B               ;BUMP LOAD POINTER LSB
0C17 78                 MOV     A,B             ;GET SET TO TEST
0C18 BD                 CMP     L               ;LOAD=CONSOLE?
0C19 CAE40A             JZ      BUFFUL          ;YES, BUFFER FULL
0C1C 15                 DCR     D               ;DEC COUNTER
0C1D C2160C             JNZ     RXI1            ;NOT DONE, TRY AGAIN
0C20 1605               MVI     D,05H           ;RESET COUNTER
0C22 E1                 POP     H               ;RESTORE LOAD POINTER
0C23 DB90       RXI2:   IN      STAT73          ;READ STATUS
0C25 E608               ANI     RXINT           ;TEST RX INT BIT
0C27 CA390C             JZ      RXI3            ;DONE, GO FINISH UP
0C2A DB90               IN      STAT73          ;READ STATUS AGAIN
0C2C E602               ANI     RXIRA           ;IS RESULT READY?
0C2E CA230C             JZ      RXI2            ;NO, TEST AGAIN
0C31 DB93               IN      RXIR73          ;YES, READ RESULT
0C33 77                 MOV     M,A             ;STORE IN BUFFER
0C34 2C                 INR     L               ;INC BUFFER POINTER
0C35 15                 DCR     D               ;DEC COUNTER
0C36 C3230C             JMP     RXI2            ;GET MORE RESULTS
0C39 7A         RXI3:   MOV     A,D             ;GET SET TO TEST
0C3A A7                 ANA     A               ;ALL RESULTS?
0C3B CA450C             JZ      RXI4            ;YES, SO FINISH UP
0C3E 3600               MVI     M,00H           ;NO, LOAD 0 TIL DONE
0C40 2C                 INR     L               ;BUMP POINTER
0C41 15                 DCR     D               ;DEC COUNTER
0C42 C3390C             JMP     RXI3            ;GO AGAIN
0C45 221020     RXI4:   SHLD    LDADR           ;UPDATE LOAD POINTER
0C48 3A1520             LDA     PRMPT           ;GET MODE INDICATOR
0C4B FE2D               CPI     '-'             ;NORMAL MODE?
0C4D CA850C             JZ      RXI6            ;YES, CLEAN UP BEFORE RETURN
                ;
                ;       POLL MODE SO CHECK CONTROL BYTE
                ;       IF CONTROL IS A POLL, SET UP SPECIAL TX COMMAND BUFFER
                ;       AND RETURN WITH POLL INDICATOR NOT 0
                ;
0C50 E1                 POP     H               ;GET PREVIOUS LOAD ADR POINTER
0C51 7E                 MOV     A,M             ;GET IC BYTE FROM BUFFER
```

611001-57

```
0C52 E61E      787          ANI    1EH       ;LOOK AT GOOD FRAME BITS
0C54 C2890C    788          JNZ    RXI5      ;IF NOT 0, INTERRUPT WASN'T FROM A GOOD FRAME
0C57 2C        789          INR    L         ;BYPASS R0 AND R1 IN BUFFER
0C58 2C        790          INR    L
0C59 2C        791          INR    L
0C5A 56        792          MOV    D,M       ;GET ADR BYTE AND SAVE IT IN D
0C5B 2C        793          INR    L
0C5C 7E        794          MOV    A,M       ;GET CNTL BYTE FROM BUFFER
0C5D FE93      795          CPI    SNRMP     ;WAS IT SNRM-P?
0C5F CA6C0C    796          JZ     T1        ;YES, GO SET RESPONSE
0C62 FE11      797          CPI    RR0P      ;WAS IT RR(0)-P?
0C64 C2890C    798          JNZ    RXI5      ;YES, GO SET RESPONSE, OTHERWISE RETURN
0C67 1E11      799          MVI    E,RR0F    ;RR(0)-P SO SET RESPONSE TO RR(0)-F
0C69 C36E0C    800          JMP    TXRET     ;GO FINISH LOADING SPECIAL BUFFER
0C6C 1E73      801 T1:      MVI    E,NSAF    ;SNRM-P SO SET RESPONSE TO NSA-F
0C6E 212020    802 TXRET:   LXI    H,CMDBF1  ;SPECIAL BUFFER ADR
0C71 36C8      806          MVI    M,0C8H    ;LOAD TX FRAME COMMAND
0C73 23        808          INX    H         ;INC POINTER
0C74 3600      809          MVI    M,00H     ;L0=0
0C76 23        810          INX    H         ;INC POINTER
0C77 3600      811          MVI    M,00H     ;L1=0
0C79 23        812          INX    H         ;INC POINTER
0C7A 72        813          MOV    M,D       ;LOAD RCVD ADR BYTE
0C7B 23        814          INX    H         ;INC POINTER
0C7C 73        815          MOV    M,E       ;LOAD RESPONSE CNTL BYTE
0C7D 3E01      816          MVI    A,01H     ;SET POLL INDICATOR NOT 0
0C7F 321620    817          STA    POLIN     ;LOAD POLL INDICATOR
0C82 C3890C    818          JMP    RXI5      ;RETURN
               819
0C85 E1        820 RXI6:    POP    H         ;CLEAN UP STACK IF NORMAL MODE
0C86 C3890C    821          JMP    RXI5      ;RETURN
               822
0C89 CD1A0B    823 RXI5:    CALL   RXDMA     ;RESET DMA CHANNEL
0C8C D1        824          POP    D         ;RESTORE REGISTERS
0C8D C1        825          POP    B
0C8E F1        826          POP    PSW
0C8F E1        827          POP    H
0C90 FB        828          EI               ;ENABLE INTERRUPTS
0C91 C9        829          RET              ;RETURN
               830 ;
               831 ;
               832 ;MESSAGE TYPER - ASSUMES MESSAGE STARTS AT HL
               833 ;
               834 ;
0C92 C5        835 TYMSG:   PUSH   B         ;SAVE BC
0C93 7E        836 TYMSG2:  MOV    A,M       ;GET ASCII CHR
0C94 23        837          INX    H         ;INC POINTER
0C95 FEFF      838          CPI    0FFH      ;STOP?
0C97 CAA10C    839          JZ     TYMSG1    ;YES, GET SET FOR EXIT
0C9A 4F        840          MOV    C,A       ;SET UP FOR DISPLAY
0C9B CDF805    841          CALL   ECHO      ;DISPLAY CHR
0C9E C3930C    842          JMP    TYMSG2    ;GET NEXT CHR
0CA1 C1        843 TYMSG1:  POP    B         ;RESTORE BC
0CA2 C9        844          RET              ;RETURN
               845 ;
               846 ;
               847 ;SIGNON MESSAGE
               848 ;
```

611001-58

```
0CA3 0D          SIGNON: DB      CR,'8273 MONITOR  V1.1',CR,0FFH
0CA4 38323733
0CA8 204D4F4E
0CAC 49544F52
0CB0 20205631
0CB4 2E31
0CB6 0D
0CB7 FF
                 ;
                 ;
                 ;
                 ;RECEIVER INTERRUPT MESSAGES
                 ;
                 ;
0CB8 0D          RXIMSG: DB      CR,'RX INT - ',0FFH
0CB9 52582049
0CBD 4E54202D
0CC1 20
0CC2 FF
                 ;
                 ;TRANSMITTER INTERRUPT MESSAGES
                 ;
0CC3 0D          TXIMSG: DB      CR,'TX INT - ',0FFH
0CC4 54582049
0CC8 4E54202D
0CCC 20
0CCD FF
                 ;
                 ;
                 ;TRANSMITTER INTERRUPT ROUTINE
                 ;
0CCE E5          TXI:    PUSH    H               ;SAVE HL
0CCF F5                  PUSH    PSW             ;SAVE PSW
0CD0 C5                  PUSH    B               ;SAVE BC
0CD1 D5                  PUSH    D               ;SAVE DE
0CD2 3E61                MVI     A,DTDMA         ;DISABLE TX DMA
0CD4 D3A8                OUT     MODE57          ;8257 MODE PORT
0CD6 1604                MVI     D,04H           ;SET COUNTER
0CD8 2A1020              LHLD    LDADR           ;GET LOAD POINTER
0CDB E5                  PUSH    H               ;SAVE IT
0CDC 45                  MOV     B,L             ;SAVE LSB IN B
0CDD 2A1320              LHLD    CNADR           ;GET CONSOLE POINTER
0CE0 04          TXI1:   INR     B               ;INC POINTER
0CE1 78                  MOV     A,B             ;GET SET TO TEST
0CE2 BD                  CMP     L               ;LOAD=CONSOLE?
0CE3 CAE40A              JZ      BUFFUL          ;YES, BUFFER FULL
0CE6 15                  DCR     D               ;NO, TEST NEXT LOCATION
0CE7 C2E00C              JNZ     TXI1            ;TRY AGAIN
0CEA E1                  POP     H               ;RESTORE LOAD POINTER
0CEB DB92                IN      TXIR73          ;READ RESULT
0CED 77                  MOV     M,A             ;STORE IN BUFFER
0CEE 2C                  INR     L               ;INR POINTER
0CEF 3600                MVI     M,00H           ;EXTRA RESULT SPOTS 0
0CF1 2C                  INR     L
0CF2 3600                MVI     M,00H
0CF4 2C                  INR     L
0CF5 3600                MVI     M,00H
0CF7 2C                  INR     L
```

611001-59

```
0CF8 3600        892        MVI    M,00H
0CFA 2C          893        INR    L
0CFB 221020      894        SHLD   LDADR          ;UPDATE LOAD POINTER
0CFE CD350B      899        CALL   TXDMA          ;RESET DMA CHANNEL
0D01 D1          900        POP    D              ;RESTORE DE
0D02 C1          901        POP    B              ;RESTORE BC
0D03 F1          902        POP    PSW            ;RESTORE PSW
0D04 E1          903        POP    H              ;RESTORE HL
0D05 FB          904        EI                    ;ENABLE INTERRUPTS
0D06 C9          905        RET                   ;RETURN
                 906 ;
                 907 ;
                 952 ;
                 953 ;
                 954        END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
ADWN   A 0922  AFCMD  A 09CE  BUFFUL A 0AE4  CH0ADR A 00A0  CH0TC  A 00A1  CH1ADR A 00A2  CH1TC  A 00A3
CMD51  A 0027  CMDBF1 A 2020  CMDBUF A 2000  CMDOUT A 0AFB  CMDREC A 0857  CMODE  A 0931  CNADR  A 2013
CNT053 A 009C  CNT153 A 009D  CNT253 A 009E  CNTL51 A 0089  CNTLC  A 0003  CNVBN  A 05BB  COBR   A 000C
COMM   A 0AE5  COMM1  A 0AEA  COMM2  A 0AFF  COMM73 A 0090  CPBF   A 0020  CR     A 000D  CRLF   A 05EB
DEM    A 0000  DEMODE A 2027  DISPY  A 0A39  DISPY1 A 0A4E  DISPY2 A 0A4D  DRDMA  A 0062  DTDMA  A 0061
ECHO   A 05F8  ENDCHK A 0A1B  ENDMA  A 0063  GDWN   A 08FF  GETCH  A 061F  GETCMD A 087D  GRCMD  A 09C4
ILLEG  A 08A7  LDADR  A 2010  LF     A 000A  LKBR1  A 2017  LKBR2  A 2018  LOOPIT A 0861  MDCNT0 A 0036
MDCNT2 A 00B6  MDE51  A 00CE  MODE53 A 009B  MODE57 A 00A8  MONTOR A 0008  NMOUT  A 06C7  NSAF   A 0073
PAR1   A 0B08  PAR2   A 0B0D  PARIN  A 0AAD  PARIN1 A 0AE0  PARIN2 A 0ADC  PARIN3 A 0ABC  PARM73 A 0091
POLIN  A 2016  PRMPT  A 2015  R0PT   A 0AA2  R1PT   A 0AA7  RBCMD  A 097B  RDCMD  A 0971  RDWN   A 08AF
RDY    A 0002  RESBUF A 2800  RESL73 A 0091  ROCMD  A 095D  RPCMD  A 09D8  RR0F   A 0011  RR0P   A 0011
RSCMD  A 0967  RST65  A 20CE  RST75  A 20D4  RXBUF  A 8200  RXD51  A 0088  RXDMA  A 0B1A  RXI    A 0C00
RXI1   A 0C16  RXI2   A 0C23  RXI3   A 0C39  RXI4   A 0C45  RXI5   A 0C89  RXI6   A 0C85  RXIMSG A 0CB8
RXINT  A 0008  RXIR73 A 0093  RXIRA  A 0002  RXS1   A 0A69  RXS2   A 0A7F  RXS3   A 0A8D  RXSORC A 0A62
RXTC   A 41FF  SBCMD  A 0985  SDWN   A 08D7  SIGNON A 0CA3  SLCMD  A 098F  SNRMP  A 0093  SOCMD  A 09A6
SPCMD  A 09E2  SRCMD  A 09BA  SSCMD  A 09B0  START  A 0800  STAT51 A 0089  STAT57 A 00A8  STAT73 A 0090
STKSRT A 20C0  SW     A 0943  T1     A 0C6C  TBUFFL A 0A24  TBUFL  A 0A07  TBUFL1 A 0A0D  TDWN   A 090E
TEST73 A 0092  TFCMD  A 09EC  TFCMD1 A 09F6  TFRET  A 0A36  TLCMD  A 0999  TRUE   A 0000  TRUE1  A 0000
TXBUF  A 8000  TXD51  A 0088  TXDMA  A 0B35  TXDMA1 A 0B42  TXI    A 0CCE  TXI1   A 0CE0  TXIMSG A 0CC3
TXINT  A 0004  TXIR73 A 0092  TXIRA  A 0001  TXPOL  A 094C  TXRET  A 0C6E  TXSORC A 0A47  TXTC   A 81FF
TYMSG  A 0C92  TYMSG1 A 0CA1  TYMSG2 A 0C93  VALDG  A 075E

ASSEMBLY COMPLETE,  NO ERRORS
```

611001-60

intel®

APPLICATION NOTE

AP-134

October 1986

Asynchronous Communication with the 8274 Multiple-Protocol Serial Controller

Order Number: 210311-002

INTRODUCTION

The 8274 Multiprotocol serial controller (MPSC) is a sophisticated dual-channel communications controller that interfaces microprocessor systems to high-speed serial data links (at speeds to 880K bits per second) using synchronous or asynchronous protocols. The 8274 interfaces easily to most common microprocessors (e.g., 8048, 8051, 8085, 8086, and 8088), to DMA controllers such as the 8237 and 8257, and to the 8089 I/O processor. Both MPSC communication channels are completely independent and can operate in a full-duplex communication mode (simultaneous data transmission and reception).

Communication Functions

The 8274 performs many communications-oriented functions, including:

— Converting data bytes from a microprocessor system into a serial bit stream for transmission over the data link to a receiving system.

— Receiving serial bit streams and reconverting the data into parallel data bytes that can easily be processed by the microprocessor system.

— Performing error checking during data transfers. Error checking functions include computing/transmitting error codes (such as parity bits or CRC bytes) and using these codes to check the validity of received data.

— Operating independently of the system processor in a manner designed to reduce the system overhead involved in data transfers.

System Interface

The MPSC system interface is extremely flexible, supporting the following data transfer modes:

1. Polled Mode. The system processor periodically reads (polls) an 8274 status register to determine when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
2. Interrupt Mode. The MPSC interrupts the system processor when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
3. DMA Mode. The MPSC automatically requests data transfers from system memory for both transmit and receive functions by means of two DMA request signals per serial channel. These DMA request signals may be directly interfaced to an 8237 or 8257 DMA controller or to an 8089 I/O processor.
4. WAIT Mode. The MPSC ready signal is used to synchronize processor data transfers by forcing the processor to enter wait states until the 8274 is ready for another data byte. This feature enables the 8274 to interface directly to an 8086 or 8088 processor by means of string I/O instructions for very high-speed data links.

Scope

This application note describes the use of the 8274 in asynchronous communication modes. Asynchronous communication is typically used to transfer data to/from video display terminals, modems, printers, and other low-to-medium-speed peripheral devices. Use of the 8274 in both interrupt-driven and polled system environments is described. Use of the DMA and WAIT modes are not described since these modes are employed mainly in synchronous communication systems where extremely high data rates are common. Programming examples are written in PL/M-86 (Appendix B and Appendix C). PL/M-86 is executed by the iAPX-86 and iAPX-88 processor families. In addition, PL/M-86 is very similar to PL/M-80 (executed by the MCS-80 and MCS-85 processor families). In addition, Appendix D describes a simple application example using an SDK-86 in an iAPX-86/88 environment.

SERIAL-ASYNCHRONOUS DATA LINKS

A serial asynchronous interface is a method of data transmission in which the receiving and transmitting systems need not be synchronized. Instead of transmitting clocking information with the data, locally generated clocks (16, 32 or 64 times as fast as the data transmission rate) are used by the transmitting and receiving systems. When a character of information is sent by the transmitting system, the character data is framed (preceded and followed) by special START and STOP bits. This framing information permits the receiving system to temporarily synchronize with the data transmission. (Refer to Figure 1 during the following discussion of asynchronous data transmission.)

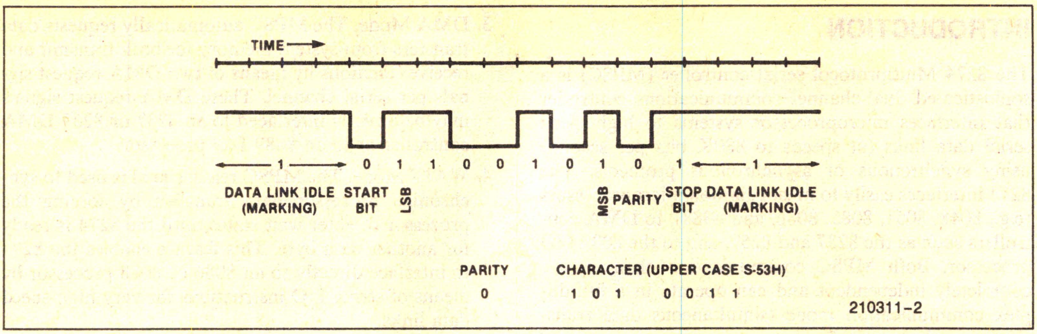


Figure 1. Transmission of a 7-Bit ASCII Character with Even Parity

Normally the data link is in an idle or marking state, continuously transmitting a "mark" (binary 1). When a character is to be sent, the character data bits are immediately preceded by a "space" (binary 0 START bit). The mark-to-space transition informs the receiving system that a character of information will immediately follow the start bit. Figure 1 illustrates the transmission of a 7-bit ASCII character (upper case S) with even parity. Note that the character is transmitted immediately following the start bit. Data bits within the character are transmitted from least-significant to most-significant. The parity bit is transmitted immediately following the character data bits and the STOP framing bit (binary 1) signifies the end of the character.

Asynchronous interfaces are often used with human interface devices such as CRT/keyboard units where the time between data transmissions is extremely variable.

Characters

In asynchronous mode, characters may vary in length from five to eight bits. The character length depends on the coding method used. For example, five-bit characters are used when transmitting Baudot Code, seven-bit characters are required for ASCII data, and eight-bit characters are needed for EBCDIC and binary data. To transmit messages composed of multiple characters, each character is framed and transmitted separately (Figure 2).

This framing method ensures that the receiving system can easily synchronize with the start and stop bits of each character, preventing receiver synchronization errors. In addition, this synchronization method makes both transmitting and receiving systems insensitive to possible time delays between character transmissions.

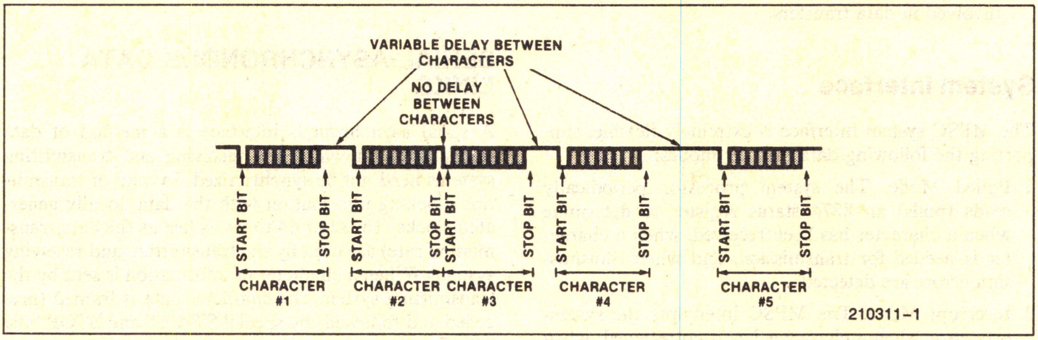


Figure 2. Multiple Character Transmission

Framing

Character framing is accomplished by the START and STOP bits described previously. When the START bit transition (mark-to-space) is detected, the receiving system assumes that a character of data will follow. In order to test this assumption (and isolate noise pulses on the data link), the receiving system waits one-half bit time and samples the data link again. If the link has returned to the marking state, noise is assumed, and the receiver waits for another START bit transition.

When a valid START bit is detected, the receiver samples the data link for each bit of the following character. Character data bits and the parity bit (if required) are sampled at their nominal centers until all required characters are received. Immediately following the data bits, the receiver samples the data link for the STOP bit, indicating the end of the character. Most systems permit specification of 1, $1\frac{1}{2}$, or 2 stop bits.

Timing

The transmitter and receiver in an asynchronous data link arrangement are clocked independently. Normally, each clock is generated locally and the clocks are not synchronized. In fact, each clock may be a slightly different frequency. (In practice, the frequency difference should not exceed a few percent. If the transmitter and receiver clock rates vary substantially, errors will occur because data bits may be incorrectly identified as START or STOP framing bits.) These clocks are designed to operate at 16, 32, or 64 times the communications data rate. These clock speeds allow the receiving device to correctly sample the incoming bit stream.

Serial-interface data rates are measured in bits/second. The term "baud" is used to specify the number of times per second that the transmitted signal level can change states. In general, the baud is not equal to the bit rate. Only when the transmitted signal has two states (electrical levels) is the baud rate equal to the bit rate. Most point-to-point serial data links use RS-232-C, RS-422, or RS-423 electrical interfaces. These specifications call for two electrical signal levels (the baud is equal to the bit rate). Modem interfaces, however, may often have differing bit and baud rates.

While there are generally no limitations on the data transmission rates used in an aysnchronous data link, a limited set of rates has been standardized to promote equipment interconnection. These rates vary from 75 bits per second to 38,400 bits per second. Table 1 illustrates typical asynchronous data rates and the associated clock frequencies required for the transmitter and receiver circuits.

Table 1. Communication Data Rates and Associated Transmitter/Receiver Clock Rates

Data Rate (Bits/Second)	Clock Rate (kHz)		
	X16	X32	X64
75	1.2	2.4	4.8
150	2.4	4.8	9.6
300	4.8	9.6	19.2
600	9.6	19.2	38.4
1200	19.2	38.4	76.8
2400	38.4	76.8	153.6
4800	76.8	153.6	307.2
9600	153.6	307.2	614.2
19200	307.2	614.4	—
38400	614.4	—	—

Parity

In order to detect transmission errors, a parity bit may be added to the character data as it is transferred over the data link. The parity bit is set or cleared to make the total number of "one" bits in the character even (even parity) or odd (odd parity). For example, the letter "A" is represented by the seven-bit ASCII code 1000001 (41H). The transmitted data code (with parity) for this character contains eight bits; 01000001 (41H) for even parity and 11000001 (OC1H) for odd parity. Note that a single bit error changes the parity of the received character and is therefore easily detected. The 8274 supports both odd and even parity checking as well as a parity disable mode to support binary data transfers.

Communication Modes

Serial data transmission between two devices can occur in one of three modes. In the simplex transmission mode, a data link can transmit data in one direction only. In the half-duplex mode, the data link can transmit data in both directions, but not simultaneously. In the full-duplex mode (the most common), the data link can transmit data in both directions simultaneously. The 8274 directly supports the full-duplex mode and will interface to simplex and half-duplex communication data links with appropriate software controls.

BREAK Condition

Asynchronous data links often include a special sequence known as a break condition. A break condition is initiated when the transmitting device forces the data link to a spacing state (binary 0) for an extended length of time (typically 150 milliseconds). Many terminals contain keys to initiate a break sequence. Under software control, the 8274 can initiate a break sequence when transmitting data and detect a break sequence when receiving data.

MPSC SYSTEM INTERFACE

Hardware Environment

The 8274 MPSC interfaces to the system processor over an 8-bit data bus. Each serial I/O channel responds to two I/O or memory addresses as shown in Table 2. In addition, the MPSC supports non-vectored and vectored interrupts.

The 8274 may be configured for memory-mapped or I/O-mapped operation.

The 8274-processor hardware interface can be configured in a flexible manner, depending on the operating mode selected—polled, interrupt-driven, DMA, or WAIT. Figure 3 illustrates typical MPSC configurations for use with an 8088 microprocessor in the polled and interrupt-driven modes.

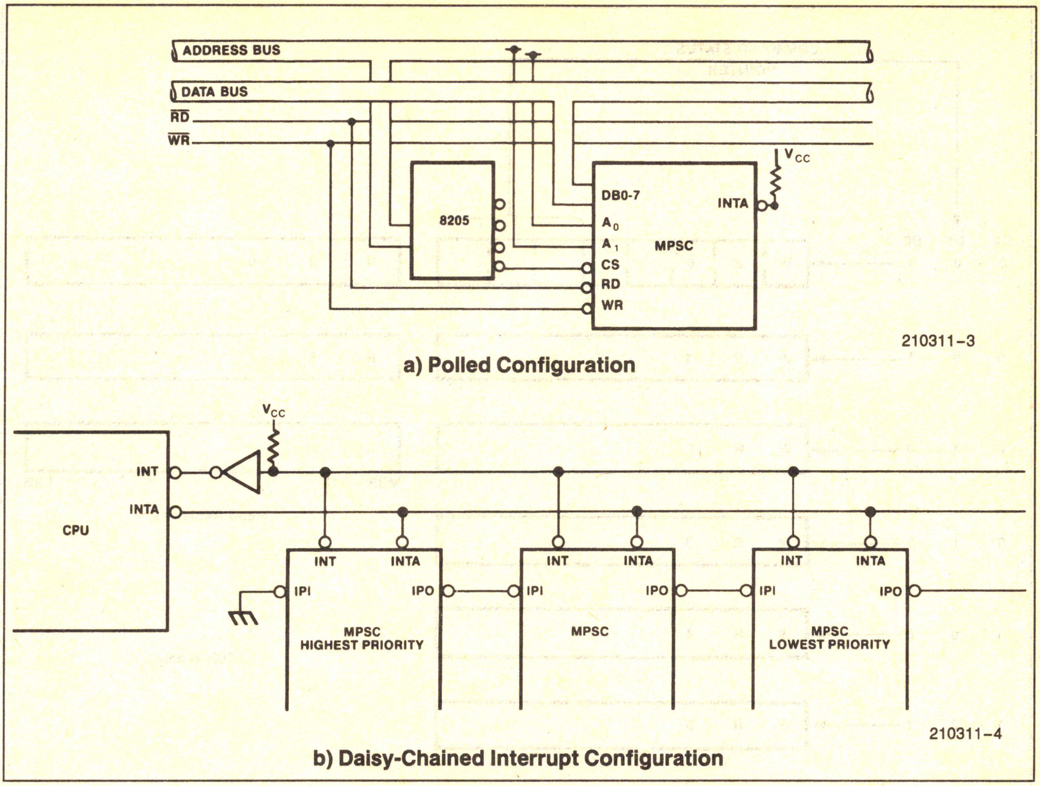
All serial-to-parallel conversion, parallel-to-serial conversion, and parity checking required during asynchronous serial I/O operation is automatically performed by the MPSC.

Operational Interface

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers and 2 readable registers for each channel, plus the interrupt vector register). These registers are all accessed by means of the command/status ports for each channel. An internal pointer register selects which of the command or status registers will be written or read during a command/status access of an MPSC channel. Figure 4 diagrams the command/status register architecture for each serial channel. In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.

Table 2. 8274 Addressing

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedance	High Impedance



a) Polled Configuration

b) Daisy-Chained Interrupt Configuration

Figure 3. 8274 Hardware Interface for Polled and Interrupt-Driven Environments

The least-significant three bits of WR0 are automatically loaded into the pointer register every time WR0 is written. After reset, WR0 is set to zero so that the first write to a command register causes the data to be loaded into WR0 (thereby setting the pointer register). After WR0 is written, the following read or write accesses the register selected by the pointer. The pointer is reset after the read or write operation is completed. In this manner, reading or writing an arbitrary MPSC channel register requires two I/O accesses. The first access is always a write command. This write command is used to set the pointer register. The second access is either a read or a write command; the pointer register (previously set) will ensure that the correct internal register is read or written. After this second access, the pointer register is automatically reset. Note that writing WR0 and reading RR0 does not require presetting of the pointer register.

During initialization and normal MPSC operation, various registers are read and/or written by the system processor. These actions are discussed in detail in the following paragraphs. Note that WR6 and WR7 are not used in the asynchronous communication modes.

RESET

When the 8274 $\overline{\text{RESET}}$ line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ($\overline{\text{RTS}}$, $\overline{\text{DTR}}$) are forced high. In addition, the pointer register is set to zero.

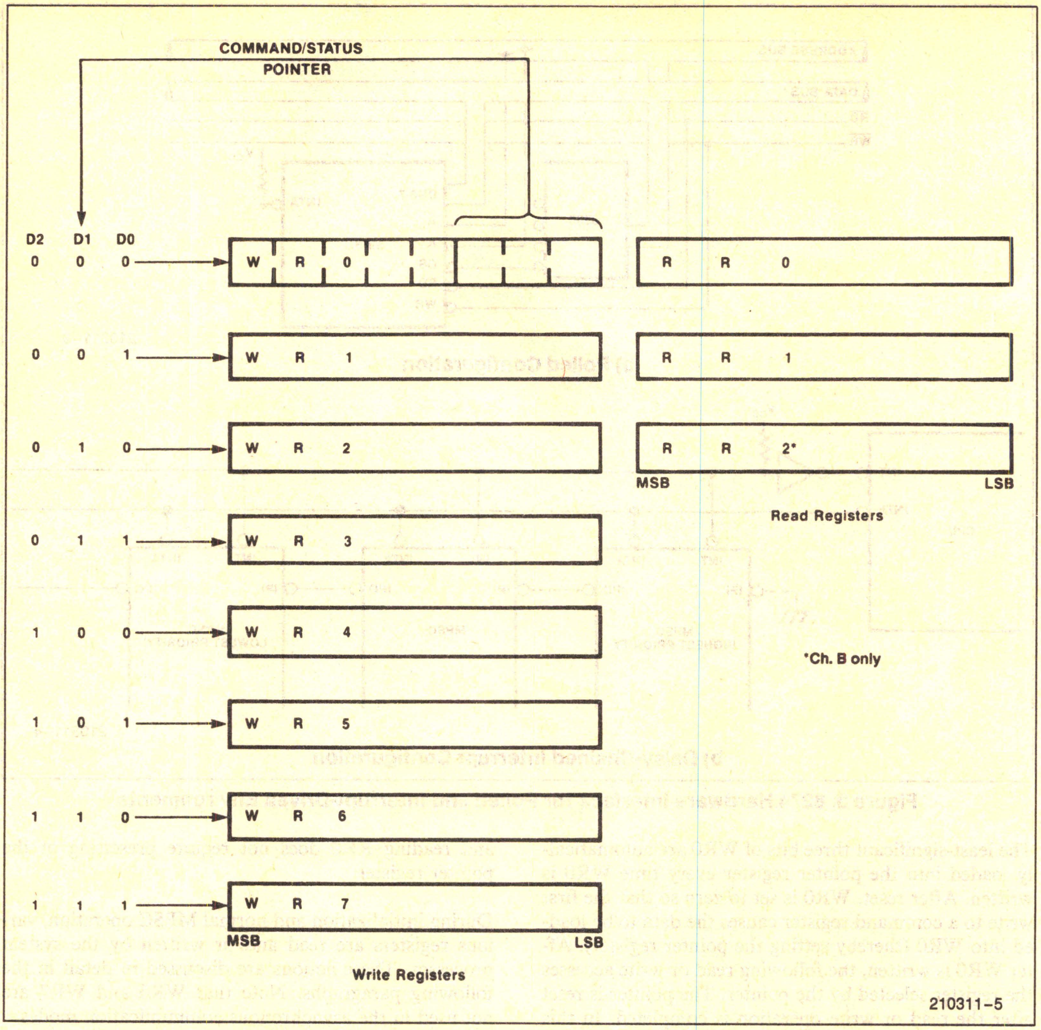


Figure 4. Command/Status Register Architecture (Each Serial Channel)

External/Status Latches

The MPSC continuously monitors the state of four external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).

A change of state in any of these monitored conditions will cause the associated status bit in RR0 (Appendix A) to be latched (and optionally cause an interrupt).

Error Reporting

Three error conditions may be encountered during data reception in the asynchronous mode:

1. Parity. If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4).

2. Framing. A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled).
3. Overrun. If an input character has been assembled but the receiver buffers are full (because the previously received characters have not been read by the system processor), an overrun error will occur. When an overrun error occurs, the input character that has just been received will overwrite the immediately preceding character.

Transmitter/Receiver Initialization

In order to operate in the asynchronous mode, each MPSC channel must be initialized with the following information:

1. Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 16, 32, or 64 times the data-link bit rate. (See Appendix A for WR4 details.)
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, $1^1/_2$, or 2. (See Appendix A for WR4 details.)
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4. (See Appendix A for WR4 details.)
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3. (See Appendix A for WR3 details.)
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3. (See Appendix A for WR3 details.)
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. (See Appendix A for WR5 details.) Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 1).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

D7	D6	D5	D4	D3	D2	D1	D0	Number of Bits Transmitted (Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5. (See Appendix A for WR5 details.)

For data transmissions via a modem or RS-232-C interface, the following information must also be specified:

1. Request-to-Send/Data-Terminal-Ready. Must be set to indicate status of data terminal equipment. Request-to-send is controlled by bit 1 of WR5 and data terminal ready is controlled by bit 7. (See Appendix A for WR5 details.)
2. Auto Enable. May be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. Auto Enable is controleld by bit 5 of WR3. (See Appendix A for WR3 details.)

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Appendix B. Figure 5 illustrates typical MPSC initialization parameters for use with this procedure.

```
call MPSC$RX$INIT(41,  1,1,0,1,  3,1,1,  3,1,1,0,1);

initializes the 8274 at address 41 as follows:

X16 clock rate          Enable transmitter
1 stop bit                and receiver
Odd parity              Auto enable set
8-bit characters        DTR and RTS set
  (Tx and Rx)           Break transmission disabled
```

Figure 5. Sample 8274 Initialization Procedure for Polled Operation

Polled Operation

In the polled mode, the processor must monitor the MPSC status by testing the appropriate bits in the read register. Data available, status, and error conditions are represented in RR0 and RR1 for channels A and B. An example of MPSC-polled transmitter/receiver routines are given in Appendix B. The following routines are detailed:

1. MPSC$POLL$RCV$CHARACTER—This procedure receives a character from the serial data link. The routine waits until the character-available flag in RR0 has been set. When this flag indicates that a character is available, RR1 is checked for errors (overrun, parity, or framing). If an error is detected, the character in the MPSC receive buffer must be read and discarded and the error routine (RECEIVE$ERROR) is called. If no receive errors have been detected, the character is input from the 8274 data port and returned to the calling program.

   MPSC$POLL$RCV$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the address of a byte variable in which to store the received character (character$ptr).

2. MPSC$POLL$TRAN$CHARACTER—This procedure transmits a character to the serial data link. The routine waits until the transmitter-buffer-empty flag has been set in RR0 before writing the character to the 8274.

   MPSC$POLL$TRAN$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the character of data that is to be transmitted (character).

3. RECEIVE$ERROR—This procedure processes receiver errors. First, an Error Reset command is written to the affected channel. All additional error processing is dependent on the specific application. For example, the receiving device may immediately request retransmission of the character or wait until a message has been completed.

   RECEIVE$ERROR requires two parameters—the address of the affected 8274 command port (cmd$port) and the error status (status) from 8274 register RR1.

Interrupt-Driven Operation

In an interrupt-driven environment, all receiver operations are reported to the system processor by means of interrupts. Once a character has been received and assembled, the MPSC interrupts the system processor. The system processor must then read the character from the MPSC data buffer and clear the current interrupt. During transmission, the system processor starts serial I/O by writing the first character of a message to the MPSC. The MPSC interrupts the system processor whenever the next character is required (i.e., when the transmitter buffer is empty) and the processor responds by writing the next character of the message to the MPSC data port for the appropriate channel.

By using interrupt-driven I/O, the MPSC proceeds independently of the system processor, signalling the processor only when characters are required for transmission, when characters are received from the data link, or when errors occur. In this manner, the system processor may continue execution of other tasks while serial I/O is performed concurrently.

Interrupt Configurations

The 8274 is designed to interface to 8085- and 8086-type processors in much the same manner as the 8259A is designed. When operating in the 8085 mode, the 8274 causes a "call" to a prespecified, interrupt-service routine location. In the 8086 mode, the 8274 presents the processor with a one-byte interrupt-type number. This interrupt-type number is used to "vector" through the 8086 interrupt service table. In either case, the interrupt service address or interrupt-type number is specified during MPSC initialization.

To shorten interrupt latency, the 8274 can be programmed to modify the prespecified interrupt vector so that no software overhead is required to determine the cause of an interrupt. When this "status affects vector" mode is enabled, the following eight interrupts are differentiated automatically by the 8274 hardware:

1. Channel B Transmitter Buffer Empty.
2. Channel B External/Status Transition.
3. Channel B Character Available.
4. Channel B Receive Error.
5. Channel A Transmitter Buffer Empty.
6. Channel A External/Status Transition.
7. Channel A Character Available.
8. Channel A Receive Error.

Interrupt Sources/Priorities

The 8274 has three interrupt sources for each channel:

1. Receiver (RxA, RxB). An interrupt is initiated when a character is available in the receiver buffer or when a receiver error (parity, framing, or overrun) is detected.
2. Transmitter (TxA, TxB). An interrupt is initiated when the transmitter buffer is empty and the 8274 is ready to accept another character for transmission.

3. External/Status (ExTA, ExTB). An interrupt is initiated when one of the external/status conditions (CDE, CTS, SYNDET, BREAK) changes state.

The 8274 supports two interrupt priority orderings (selectable during MPSC initialization) as detailed in Appendix A, WR2, CH-A.

Interrupt Initialization

In addition to the initialization parameters required for polled operation, the following parameters must be supplied to the 8274 to specify interrupt operation:

1. Transmit Interrupt Enable. Transmitter-buffer-empty interrupts are separately enabled by bit 1 of WR1. (See Appendix A for WR1 details.)
2. Receive Interrupt Enable. Receiver interrupts are separately enabled in one of three modes: a) interrupt on first received character only and on receive errors (used for message-oriented transmission systems), b) interrupt on all received characters and on receive errors, but do not interrupt on parity errors, and c) interrupt on all received characters and on receive errors (including parity errors). The ability to separately disable parity interrupts can be extremely useful when transmitting messages. Since the parity error bit in RR1 is latched, it will not be reset until an error reset operation is performed. Therefore, the parity error bit will be set if any parity errors were detected in a multi-character message. If this mode is used, the serial I/O software must poll the parity error bit at the completion of a message and issue an error reset if appropriate. The receiver interrupt mode is controlled by bits 3 and 4 of WR1. (See Appendix A for WR1 details.)
3. External/Status Interrupts. External/Status interrupts can be separately enabled by bit 0 of WR1. (See Appendix A for WR1 details.)
4. Interrupt Vector. An eight-bit interrupt-service routine location (8085) or interrupt type (8086) is specified through WR2 of channel B. (See Appendix A for WR2 details.) Table 3 lists interrupt vector addresses generated by the 8274 in the "status affects vector" mode.
5. "Status Affects Vector" Mode. The 8274 will automatically modify the interrupt vector if bit 3 of WR1 is set. (See Appendix A for WR1 details.)
6. System Configuration. Specifies the 8274 data transfer mode. Three configuration modes are available: a) interrupt-driven operation for both channels, b) DMA operation for both channels, and c) DMA operation for channel A, interrupt-driven operation for channel B. The system configuration is specified by means of bits 0 and 1 of WR2 (channel A). (See Appendix A for WR2 details.)
7. Interrupt Priorities. The 8274 permits software specification of receive/transmit priorities by means of bit 2 of WR2 (channel A). (See Appendix A for WR2 details.)
8. Interrupt Mode. Specifies whether the MPSC is to operate in a non-vectored mode (for use with an external interrupt controller), in an 8086-vectored mode, or in an 8085-vectored mode. This parameter is specified through bits 3 and 4 of WR2 (channel A). (See Appendix A for WR2 details.)

An MPSC interrupt initialization procedure (MPSCINTINIT) is listed in Appendix C.

Table 3. MPSC-Generated Interrupt Vectors in "Status Affects Vector" Mode

V7	V6	V5	V4	V3	V2	V1	V0	V7	V6	V5	V4	V3	V2	V1	V0	Original Vector (Specified during Initialization)
8086 Interrupt Type								8085 Interrupt Location								Interrupt Condition
V7	V6	V5	V4	V3	0	0	0	V7	V6	V5	0	0	0	V1	V0	Channel B Transmitter Buffer Empty
V7	V6	V5	V4	V3	0	0	1	V7	V6	V5	0	0	1	V1	V0	Channel B External/Status Change
V7	V6	V5	V4	V3	0	1	0	V7	V6	V5	0	1	0	V1	V0	Channel B Receiver Character Available
V7	V6	V5	V4	V3	0	1	1	V7	V6	V5	0	1	1	V1	V0	Channel B Receive Error
V7	V6	V5	V4	V3	1	0	0	V7	V6	V5	1	0	0	V1	V0	Channel A Transmitter Buffer Empty
V7	V6	V5	V4	V3	1	0	1	V7	V6	V5	1	0	1	V1	V0	Channel A External/Status Change
V7	V6	V5	V4	V3	1	1	0	V7	V6	V5	1	1	0	V1	V0	Channel A Receiver Character Available
V7	V6	V5	V4	V3	1	1	1	V7	V6	V5	1	1	1	V1	V0	Channel A Receive Error

Interrupt Service Routines

Appendix C lists four interrupt service procedures, a buffer transmission procedure, and a buffer reception procedure that illustrate the use of the 8274 in interrupt-driven environments. Use of these procedures assumes that the 8086/8088 interrupt vector is set to 20H and that channel B is used with the "status affects vector" mode enabled.

1. TRANSMIT$BUFFER—This procedure begins serial transmission of a data buffer. Two parameters are required—a pointer to the buffer (buf$ptr) and the length of the buffer (buf$length). The procedure first sets the global buffer pointer, buffer length, and initial index for the transmitter-interrupt service routine and initiates transmission by writing the first character of the buffer to the 8274. The procedure then enters a wait loop until the I/O completion status is set by the transmit-interrupt service routine (MPSC$TRANSMIT$CHARACTER$INT).
2. RECEIVE$BUFFER—This procedure inputs a line (terminated by a line feed) from a serial I/O port. Two parameters are required—a pointer to the input buffer (buf$ptr) and a pointer to the buffer length variable (buf$length$ptr). The buffer length will be set by this procedure when the complete line has been input. The procedure first sets the global buffer pointer and initial index for the receiver interrupt service routine. RECEIVE$BUFFER then enters a wait loop until the I/O completion status is set by the receive interrupt routine (MPSC$RECEIVE-$CHARACTER$INT).
3. MPSC$TRANSMIT$CHARACTER$INT—This procedure is executed when the MPSC Tx-buffer-empty interrupt is acknowledged. If the current transmit buffer index is less than the buffer length, the next character in the buffer is written to the MPSC data port and the buffer pointer is updated. Otherwise, the transmission complete status is posted.
4. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when a character has been assembled by the MPSC and the MPSC has issued a character-available interrupt. If no input buffer has been set up by RECEIVE$BUFFER, the character is ignored. If a buffer has been set up, but it is full, a receive overrun error is posted. Otherwise, the received character is read from the MPSC data port and the buffer index is updated. Finally, if the received character is a line feed, the reception complete status is posted.
5. RECEIVE$ERROR$INT—This procedure is executed when a receive error is detected. First, the error conditions are read from RR1 and the character currently in the MPSC receive buffer is read and discarded. Next, an Error Reset command is written to the affected channel. All additional error processing is application dependent.
6. EXTERNAL$STATUS$CHANGE$INT—This procedure is executed when an external status condition change is detected. The status conditions are read from RR0 and a Reset External/Status Interrupt command is issued. Further error processing is application dependent.

DATA LINK INTERFACE

Serial Data Interface

Each serial I/O channel within the 8274 MPSC interfaces to two data link lines—one line for transmitting data and one for receiving data. During transmission, characters are converted from parallel data format (as supplied by the system processor or DMA device) into a serial bit stream (with START and STOP bits) and clocked out on the TxD pin. During reception, a serial bit stream is input on the RxD pin, framing bits are stripped out of the data stream, and the resulting character is converted to parallel data format and passed to the system processor or DMA device.

Data Clocking

As discussed previously, the frequency of data transmission/reception on the data link is controlled by the MPSC clock in conjunction with the programmed clock divider (in register WR4). The 8274 is designed to permit all four serial interface lines (TxD and RxD for each channel) to operate at different data rates. Four clock input pins (TxC and RxC for each channel) are available for this function. Note that the clock rate divider spcified in WR4 is used for both RxC and TxC on the appropriate channel; clock rate dividers for each channel are independent.

Modem Control

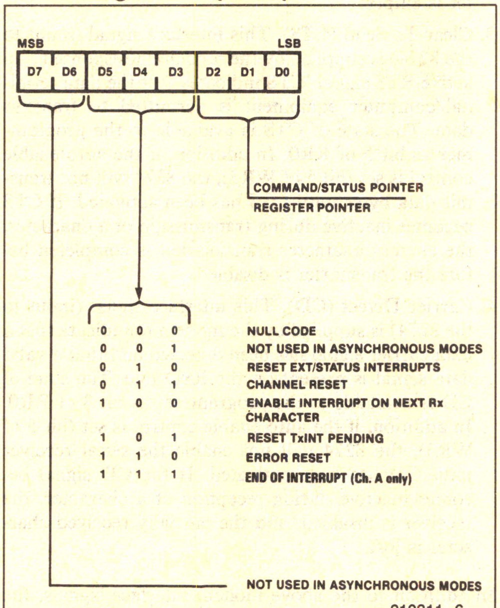
The following four modem interface signals may be connected to the 8274:

1. Data Terminal Ready (DTR). This interface signal (output by the 8274) is software controlled through bit 7 of WR5. When active, DTR indicates that the data terminal/computer equipment is active and ready to interact with the data communications channel. In addition, this signal prepares the modem for connection to the communication channel and maintains connections previously established (e.g., manual call origination).
2. Request To Send (RTS). This interface signal (output by the 8274) is software controlled through bit 1 of WR5. When active, RTS indicates that the data terminal/computer equipment is ready to transmit data. When the RTS bit is reset in asynchronous mode, the signal does not go high until the transmitter is empty.
3. Clear To Send (CTS). This interface signal (input to the 8274) is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit data. The state of CTS is available to the programmer as bit 5 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not transmit data bytes until CTS has been activated. If CTS becomes inactive during transmission of a character, the current character transmission is completed before the transmitter is disabled.
4. Carrier Detect (CD). This interface signal (input to the 8274) is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD line. The state of CD is available to the programmer as bit 3 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not enable the serial receiver until CD has been activated. If the CD signal becomes inactive during reception of a character, the receiver is disabled, and the partially received character is lost.

In addition to the above modem interface signals, the 8274 SYNDET input pin for channel A may be used as a general-purpose input in the asynchronous communication mode. The status of this signal is available to the programmer as bit 4 of status register RR0.

APPENDIX A
COMMAND/STATUS DETAILS FOR ASYNCHRONOUS COMMUNICATION

Write Register 0 (WR0):



210311-6

D2,D1,D0 Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5,D4,D3 Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Note used in asynchronous modes.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and reenables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt-on-First-Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt Pending—if the Transmit Interrupt mode is selected, the MPSC automatically interrupts data when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

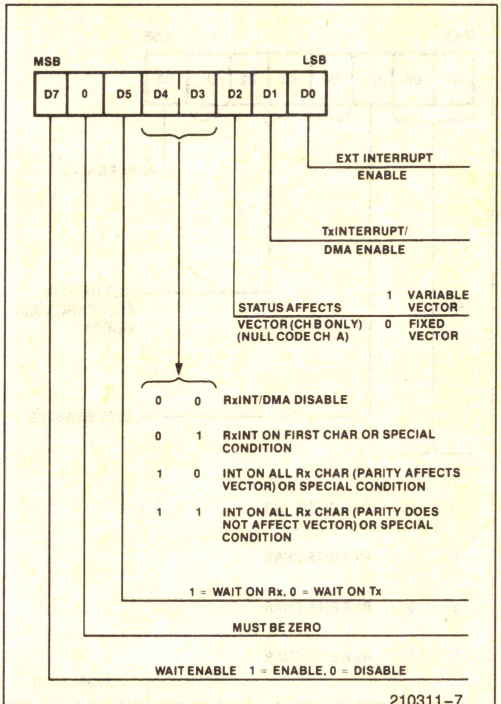
Write Register 1 (WR1):

D0 External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{\text{CD}}$, $\overline{\text{CTS}}$ or $\overline{\text{SYNDET}}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects Vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set, then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

Write Register 1 (WR1):



210311-7

D4,D3 Receive Interrupt Mode.

0 0 Receive Interrupts/DMA Disabled.

0 1 Receive Interrupt on First Character Only or Special Condition.

1 0 Interrupt on All Receive Characters of Special Condition (Parity Error is a Special Receive Condition).

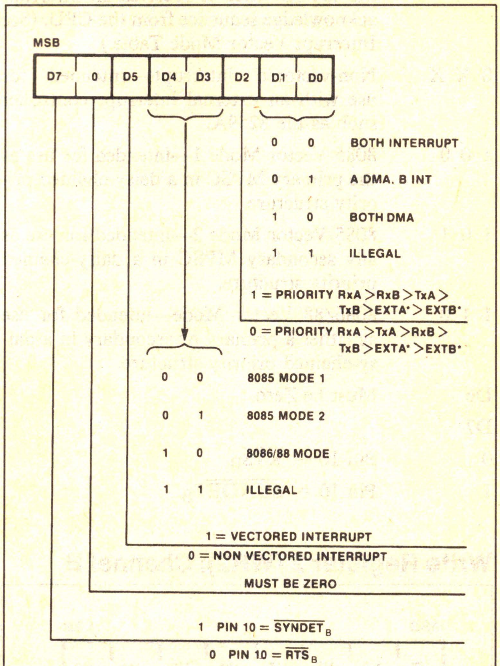
1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/Transmit—when the following conditions are met, the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}$ = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired *or* connected since only one signal is active at any one time while the other is in the High Z state.

D6 Must be Zero.

D7 Wait Enable—enables the wait function.

Write Register 2 (WR2): Channel A



NOTE: 210311-8

*External Status Interrupt—only if EXT Interrupt Enable (WR1; D0) is set.

D1,D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 Channel A and Channel B both use interrupts.

0 1 Channel A uses DMA, Channel B uses interrupt.

1 0 Channel A and Channel B both use DMA.

1 1 Illegal Code.

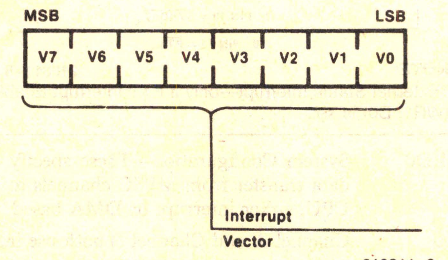
D2 Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

0 (Highest) RxA, TxA, RxA, RxB, TxBExTA, ExTB (Lowest).

1 (Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).

D5,D4,D3	Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table.)
0 X X	Non-vectored interrupts—intended for use with an external interrupt controller such as the 8259A.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy-chained priority structure.
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy-chained priority structure.
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy-chained priority structure.
D6	Must be Zero.
D7	
0	Pin 10 = $\overline{RTS}_B$.
1	Pin 10 = $\overline{SYNDET}_B$.

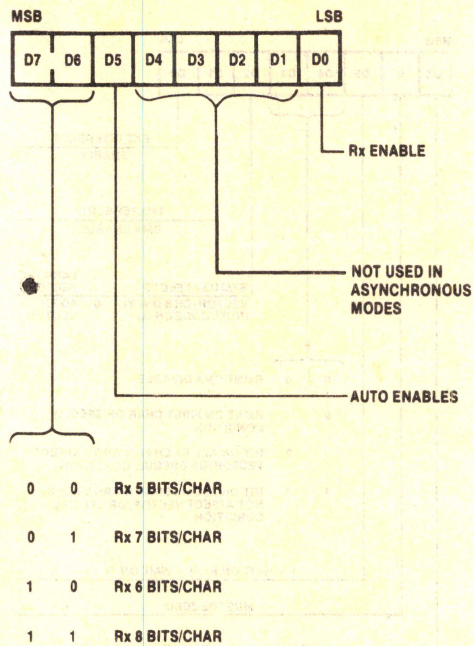
Write Register 2 (WR2): Channel B



210311-9

D7–D0	Interrupt vector—this register contains the value of the interrupt vector placed on the data bus during acknowledge sequences.

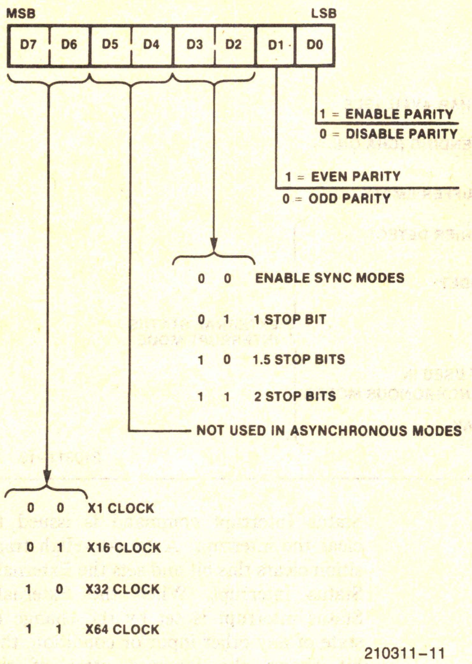
Write Register 3 (WR3):



210311-10

D0	Receiver Enable—a one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D5	Auto Enables—a one written to this bit causes $\overline{CD}$ to be an automatic enable signal for the receiver and $\overline{CTS}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).
D7,D6	Receiver Character length.
0 0	Receive 5 Data bits/character.
0 1	Receive 7 Data bits/character.
1 0	Receive 6 Data bits/character.
1 1	Receive 8 Data bits/character.

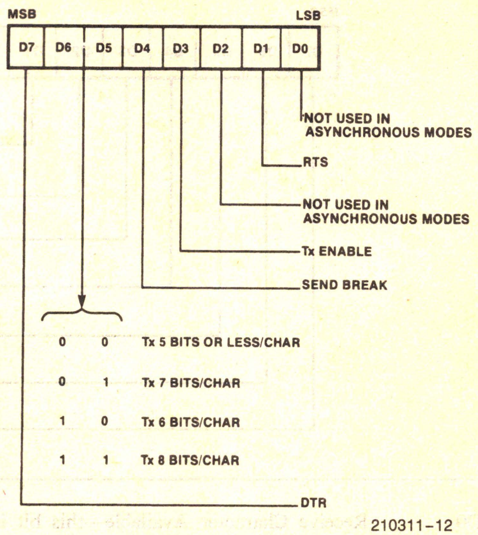
Write Register 4 (WR4):



210311-11

D0	Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.
D1	Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and zero causes it to send and expect odd parity.
D3,D2	Stop Bits.
0 0	Selects synchronous modes.
0 1	Async mode, 1 stop bit/character.
1 0	Async mode, 1½ stop bits/character.
1 1	Async mode, 2 stop bits/character.
D7,D6	Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. If the 1x mode is selected, bit synchronization must be done externally.
0 0	Clock rate = Data rate × 1.
0 1	Clock rate = Data rate × 16.
1 0	Clock rate = Data rate × 32.
1 1	Clock rate = Data rate × 64.

Write Register 5 (WR5):



210311-12

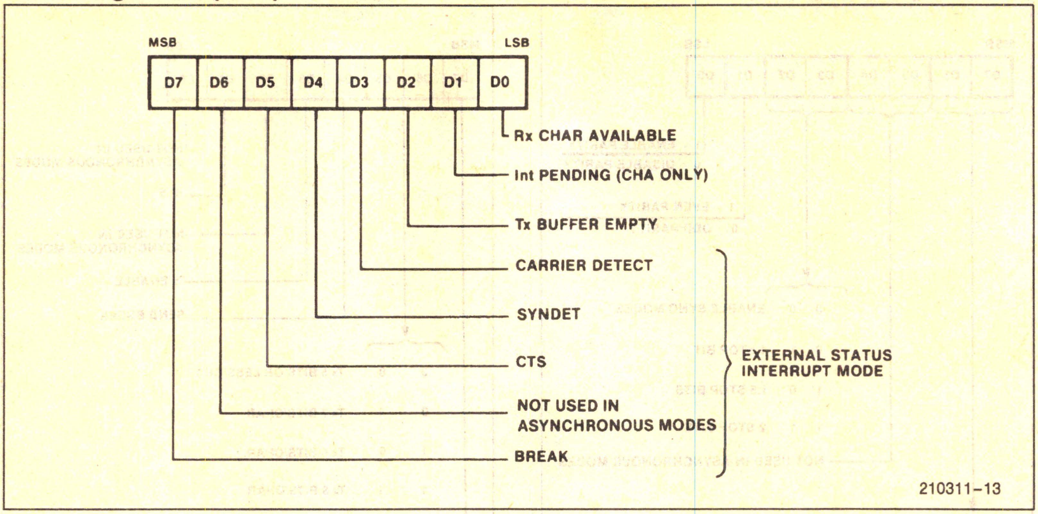
D1	Request to Send—a one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high). When the RTS bit is reset in asynchronous mode, the signal does not go inactive until the transmitter is empty.
D3	Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.
D4	Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.
D6,D5	Transmit Character length.
0 0	Transmit 5 or less bits/character.
0 1	Transmit 7 bits/character.
1 0	Transmit 6 bits/character.
1 1	Transmit 8 bits/character.

Bits to be sent must be right justified, least-significant bit first, e.g.:

D7 D6 D5 D4 D3 D2 D1 D0

0 0 B5 B4 B3 B2 B1 B0

Read Register 0 (RR0):



D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending—This Interrupt-Pending bit is reset when an E01 command is issued and there is no other interrupt request pending at that time. In vector mode, this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

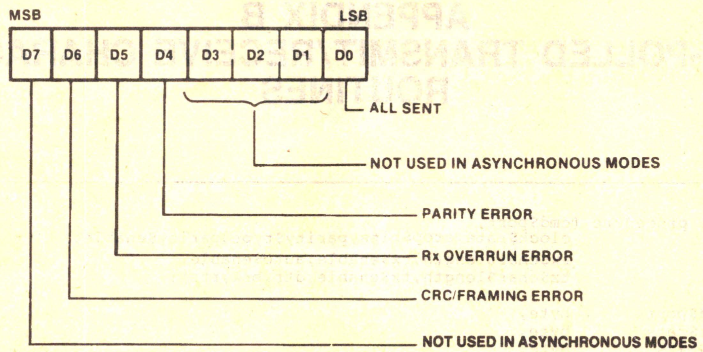
D3 Carrier Detect—This bit contains the state of the $\overline{CD}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CD}$ pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{CD}$ pin immediately following a Reset External/Status Interrupt command.

D4 SYNDET—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that it shows the state of the $\overline{SYNDET}$ input. Any High-to-Low transition on the $\overline{SYNDET}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{SYNDET}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNDET}$ input.

D5 Clear to Send—this bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D7 Break—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

Read Register 1 (RR1):



210311-14

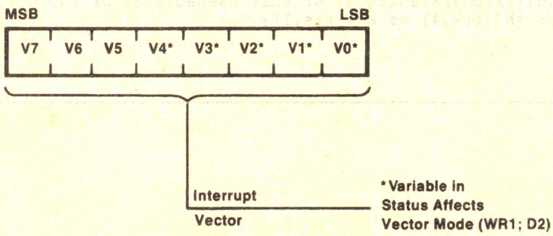
The Break bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single, extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

D0 All sent—this bit is set when all characters have been sent. It is reset when characters are in the transmitter. In synchronous modes, this bit is always set.

D4 Parity Error—if parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5 Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwrittenand flagged with this error. Once the overwritten character is read, this error condition is latched until

Read Register 2 (RR2):



210311-15

reset by the Error Reset command. If the MPSC is in the "status affects vector" mode, the overrun causes a Special Receive Error Vector.

D6 Framing Error—in async modes, a one in this bit indicates a receive framing error. It can be reset by issuing an Error Reset command.

RR2 Channel B

D7–D0 Interrupt vector—contains the interrupt vector programmed into WR2. If the "status affects vector" mode is selected, it contains the modified vector. (See WR2.) RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one. May be read from Channel B only.

APPENDIX B
MPSC-POLLED TRANSMIT/RECEIVE CHARACTER ROUTINES

```
MPSC$RX$INIT: procedure (cmd$port,
                         clock$rate,stop$bits,parity$type,parity$enable,
                         rx$char$length,rx$enable,auto$enable,
                         tx$char$length,tx$enable,dtr,brk,rts);

    declare cmd$port         byte,
            clock$rate       byte,
            stop$bits        byte,
            parity$type      byte,
            parity$enable    byte,
            rx$char$length   byte,
            rx$enable        byte,
            auto$enable      byte,
            tx$char$length   byte,
            tx$enable        byte,
            dtr              byte,
            brk              byte,
            rts              byte;


    output(cmd$port)=30H;           /* channel reset */

    output(cmd$port)=14H;           /* point to WR4 */
    /* set clock rate, stop bits, and parity information */
    output(cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                     or parity$enable;

    output(cmd$port)=13H;           /* point to WR3 */
    /* set up receiver parameters */
    output(cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

    output(cmd$port)=15H;           /* point to WR5 */
    /* set up transmitter parameters */
    output(cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                     or shl(brk,4) or shl(rts,1);

    end MPSC$RX$INIT;
```

210311-16

```
MPSC$POLL$RCV$CHARACTER: procedure(data$port,cmd$port,character$ptr) byte;

  declare data$port      byte,
          cmd$port       byte,
          character$ptr  pointer,
          character      based character$ptr byte,
          status         byte;

  declare char$avail     literally '1',
          rcv$error      literally '70H';

  /* wait for input character ready */
  while (input(cmd$port) and char$avail) <> 0 do; end;

  /* check for errors in received character */
  output(cmd$port)=1;                           /* point to RR1 */
  if (status:=input(cmd$port) and rcv$error)
    then do;
      character=input(data$port);               /* read character to clear MPSC */
      call RECEIVE$ERROR(cmd$port,status);      /* clear receiver errors */
      return 0;                                 /* error return - no character avail */
      end;
    else do;
      character=input(data$port);
      return 0FFH;                               /* good return - character avail */
      end;

end MPSC$POLL$RCV$CHARACTER;


MPSC$POLL$TRAN$CHARACTER: procedure(data$port,cmd$port,character);

  declare data$port      byte,
          cmd$port       byte,
          character      byte;

  declare tx$buffer$empty literally '4';

  /* wait for transmitter buffer empty */
  while not (input(cmd$port) and tx$buffer$empty) do; end;

  /* output character */
  output(data$port)=character;

end MPSC$POLL$TRAN$CHARACTER;


RECEIVE$ERROR: procedure(cmd$port,status);

  declare cmd$port       byte,
          status         byte;

  output(cmd$port)=30H;              /* error reset */

  /* *** other application dependent
         error processing should be placed here    *** */

end RECEIVE$ERROR;
```

210311–17

```
TRANSMIT$BUFFER: procedure(buf$ptr,buf$length)

  declare
    buf$ptr        pointer,
    buf$length     byte;

  /* set up transmit buffer pointer and buffer length in global variables for
     interrupt service */
  tx$buffer$ptr=buf$ptr;
  transmit$length=buf$length;

  transmit$status=not$complete;           /* setup status for not complete */
  output(data$port)=transmit$buffer(0);   /* transmit first character */
  transmit$index=1;                       /* first character transmitted */

  /* wait until transmission complete or error detected */
  while transmit$status = not$complete do; end;
  if transmit$status <> complete
    then return false;
    else return true;

end TRANSMIT$BUFFER;



RECEIVE$BUFFER: procedure (buf$ptr,buf$length$ptr);

  declare
    buf$ptr          pointer,
    buf$length$ptr   pointer,
    buf$length       based buf$length$ptr byte;

  /* set up receive buffer pointer in global variable for interrupt service */
  rx$buffer$ptr=buf$ptr;
  receive$index=0;

  receive$status=not$complete;        /* set status to not complete */
  /* wait until buffer received */
  while receive$status = not$complete do; end;
  buf$length=receive$length;
  if receive$status = complete
    then return true;
    else return false;

end RECEIVE$BUFFER;
```

210311-18

APPENDIX C
INTERRUPT-DRIVEN TRANSMIT/RECEIVE SOFTWARE

```
declare
  /* global variables for buffer manipulation */

  rx$buffer$ptr           pointer,              /* pointer to receive buffer */
  receive$buffer based rx$buffer$ptr(128) byte,
  receive$status          byte initial(0),      /* indicates receive buffer status */
  receive$index           byte,                 /* current index into receive buffer */
  receive$length          byte,                 /* length of final receive buffer */

  tx$buffer$ptr           pointer,              /* pointer to transmit buffer */
  transmit$buffer based tx$buffer$ptr(128) byte,
  transmit$status         byte initial(0),      /* indicates transmit buffer status */
  transmit$index          byte,                 /* current index into transmit buffer */
  transmit$length         byte,                 /* length of buffer to be transmitted */

  cmd$port                literally '43H',
  data$port               literally '41H',
  a$cmd$port              literally '42H',
  b$cmd$port              literally '43H',
  line$feed               literally '0AH',
  not$complete            literally '0',
  complete                literally '0FFH',
  overrun                 literally '1',

  channel$reset           literally '18H',
  error$reset             literally '30H',
  reset$ext$status        literally '10H';
```

210311-20

```
MPSC$INT$INIT: procedure (clock$rate,stop$bits,parity$type,parity$enable,
                          rx$char$length,rx$enable,auto$enable,
                          tx$char$length,tx$enable,dtr,brk,rts,
                          ext$en,tx$en,rx$en,stat$affects$vector,
                          config,priority,vector$int$mode,int$vector);

  declare
        clock$rate        byte,         /* 2-bit code for clock rate divisor */
        stop$bits         byte,         /* 2-bit code for number of stop bits */
        parity$type       byte,         /* 1-bit parity type */
        parity$enable     byte,         /* 1-bit parity enable */
        rx$char$length    byte,         /* 2-bit receive character length */
        rx$enable         byte,         /* 1-bit receiver enable */
        auto$enable       byte,         /* 1-bit auto enable flag */
        tx$char$length    byte,         /* 2-bit transmit character length */
        tx$enable         byte,         /* 1-bit transmitter enable */
        dtr               byte,         /* 1-bit status of DTR pin */
        brk               byte,         /* 1-bit data link break enable */
        rts               byte,         /* 1-bit status of RTS pin */
        ext$en            byte,         /* 1-bit external/status enable */
        tx$en             byte,         /* 1-bit Tx interrupt enable */
        rx$en             byte,         /* 2-bit Rx interrupt enable/mode */
        stat$aff$vector   byte,         /* 1-bit status affects vector flag */
        config            byte,         /* 2-bit system config - int/DMA */
        priority          byte,         /* 1-bit priority flag */
        vector$int$mode   byte,         /* 3-bit interrupt mode code */
        int$vector        byte;         /* 8-bit interrupt type code */


  output(b$cmd$port)=channel$reset;    /* channel reset */

  output(b$cmd$port)=14H;              /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(b$cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                    or parity$enable;

  output(b$cmd$port)=13H;              /* point to WR3 */
  /* set up receiver parameters */
  output(b$cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(b$cmd$port)=15H;           /* point to WR5 */
  /* set up transmitter parameters */
  output(b$cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                  or shl(brk,4) or shl(rts,1);

  output(b$cmd$port)=12H;                /* point to WR2 */
  /* set up interrupt vector */
  output(b$cmd$port)=int$vector;

  output(a$cmd$port)=12H;                /* point to WR2, channel A */
  /* set up interrupt modes */
  output(a$cmd$port)=shl(vector$int$mode,3) or shl(priority,2) or config;

  output(b$cmd$port)=11H;                /* point to WR1 */
  /* set up interrupt enables */
  output(b$cmd$port)=shl(rx$en,3) or shl(stat$aff$vector,2) or shl(tx$en,1)
                    or ext$en;

  end MPSC$INT$INIT;
```

210311-21

```
MPSC$RECEIVE$CHARACTER$INT: procedure interrupt 22H;

   /* ignore input if no open buffer */
   if receive$status <> not$complete then return;

   /* check for receive buffer overrun */
   if receive$index = 128
     then receive$status=overrun;
     else do;
       /* read character from MPSC and place in buffer - note that the
          parity of the character must be masked off during this step if
          the character is less than 8 bits (e.g., ASCII)  */
       receive$buffer(receive$index),character=input(data$port) and 7FH;
       receive$index=receive$index+1;       /* update receive buffer index */

       /* check for line feed to end line */
       if character = line$feed
         then do; receive$length=receive$index; receive$status=complete; end;
     end;

end MPSC$RECEIVE$CHARACTER$INT;


MPSC$TRANSMIT$CHARACTER$INT: procedure interrupt 20H;

   /* check for more characters to transfer */
   if transmit$index < transmit$length
     then do;
       /* write next character from buffer to MPSC */
       output(data$port)=transmit$buffer(transmit$index);
       transmit$index=transmit$index+1;     /* update transmit buffer index */
     end;
     else transmit$status=complete;

end MPSC$TRANSMIT$CHARACTER$INT;

RECEIVE$ERROR$INT: procedure interrupt 23H;

   declare
     temp                      byte;     /* temporary character storage */

   output(cmd$port)=1;                   /* point to RR1 */
   receive$status=input(cmd$port);
   temp=input(data$port);                /* discard character */
   output(cmd$port)=error$reset;         /* send error reset */

   /* *** other application dependent
          error processing should be placed here   *** */

end RECEIVE$ERROR$INT;


EXTERNAL$STATUS$CHANGE$INT: procedure interrupt 21H;

   transmit$status=input(cmd$port)       /* input status change information */
   output(cmd$port)=reset$ext$status;

   /* *** other application dependent
          error processing should be placed here   *** */

end EXTERNAL$STATUS$CHANGE$INT;
```

210311-19

APPENDIX D
APPLICATION EXAMPLE USING SDK-86

This application example shows the 8274 in a simple iAPX-86/88 system. The 8274 controls two separate asynchronous channels using its internal interrupt controller to request all data transfers. The 8274 driver software is described which transmits and receives data buffers provided by the CPU. Also, status registers are maintained in system memory to allow the CPU to monitor progress of the buffers and error conditions.

THE HARDWARE INTERFACE

Nothing could be easier than the hardware design of an interrupt-driven 8274 system. Simply connect the data bus lines, a few bus control lines, supply a timing clock for baud rate and, voila, it's done! For this example, the ubiquitous SDK-86 is used as the host CPU system. The 8274 interface is constructed on the wire-wrap area provided. While discussing the hardware interface, please refer to Diagram 1.

Placing the 8274 on the lower 8 bits of the 8086 data bus allows byte-wide data transfers at even I/O addresses. For simplicity, the 8274's $\overline{CS}$ input is generated by combining the M/IO select line with address line A7 via a 7432. This places the 8274 address range in multiple spots within the 8086 I/O address space. (While fine for this example, a more complete address decoding is recommended for actual prototype systems.) The 8086's A1 and A2 address lines are connected to the A0 and A1 8274 register select inputs respectively. Although other port assignments are possible because of the overlapping address spaces, the following I/O port assignments are used in this example:

Port Function	I/O Address
Data channel A	0000H
Command/status A	0002H
Data channel B	0004H
Command/status B	0006H

To connect the 8274's interrupt controller into the system an inverter and pull-up resistor are needed to convert the 8274's active-low, interrupt-request output, $\overline{INT}$, into the correct polarity for the 8086's INTR interrupt input. The 8274 recognizes interrupt-acknowledge bus cycles by connecting the $\overline{INTA}$ (INTerrupt Acknowledge) lines of the 8274 and 8086 together.

The 8274 ReaD and WRite lines directly connect to the respective 8086 lines. The RESET line requires an inverter. The system clock for the 8274 is provided by the PCLK (peripheral clock) output of the 8284A clock generator.

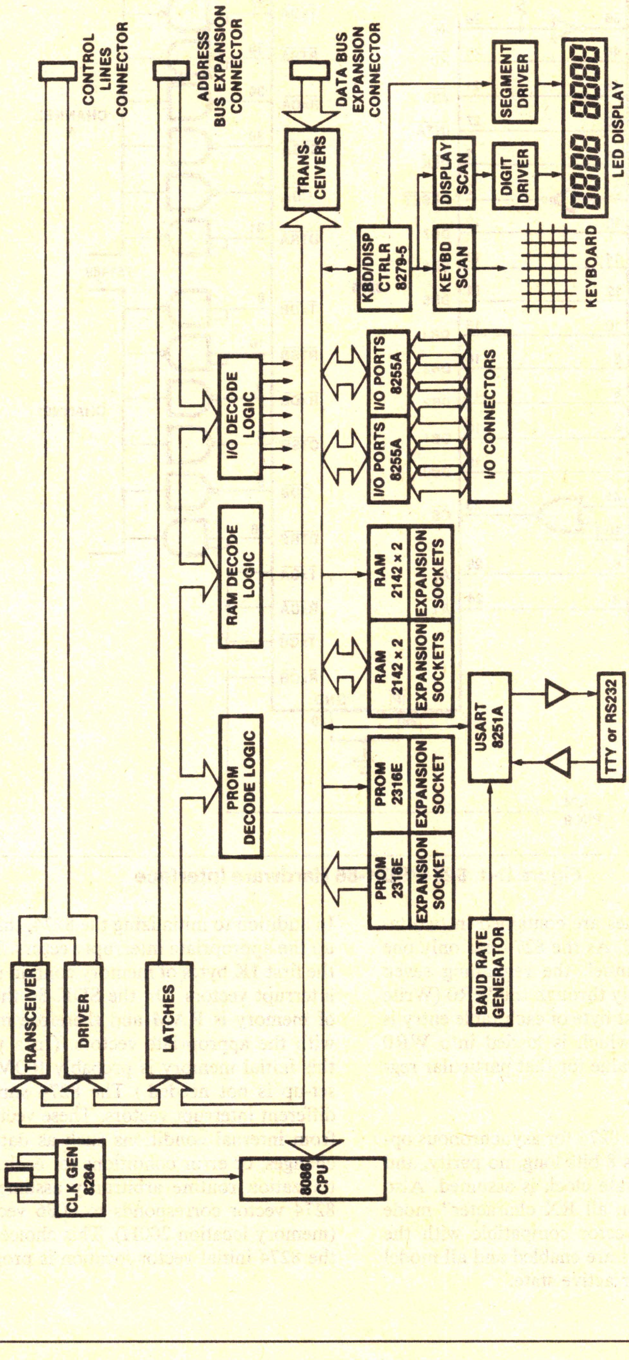
On the 8274's serial side, traditional 1488 and 1489 RS-232 drivers and receivers are used for the serial interface. The onboard baud rate generator supplies the channel baud rate timing. In this example, both sides of both channels operate at the same baud rate although this certainly is not a requirement. (On the SDK-86, the baud rate selection is hard-wired thru jumpers. A more flexible approach would be to incorporate an 8253 Programmable Interval Timer to allow software-configurable baud rate selection.)

That's all there is to it. This hardware interface is completely general-purpose and supports all of the 8274 features except the DMA data transfer mode which requires an external DMA controller. Now let's look at the software interface.

SOFTWARE INTERFACE

In this example, it is assumed that the 8086 has better things to do rather than continuously run a serial channel. Presenting the software as a group of callable procedures lets the designer include them in the main body of another program. The interrupt-driven data transfers give the effect that the serial channels are handled in the background while the main program is executing in the foreground. There are five basic procedures: a serial channel initialization routine and buffer handling routines for the transmit and receive data buffers of each channel. Appendix D-1 shows the entire software listing. Listing line numbers are referenced as each major routing is discussed.

The channel initialization routine (INITIAL 8274), starting with line #203, simply sets each channel into a particular operating mode by loading the command registers of the 8274. In normal operation, once these registers are loaded, they are rarely changed. (Although this example assumes a simple asynchronous operating mode, the concept is easily extended for the byte- and bit-synchronous modes.)



(For detailed description on SDK-86, refer to SDK-86 MCS-86 System Design Kit Assembly Manual.)

210311-22

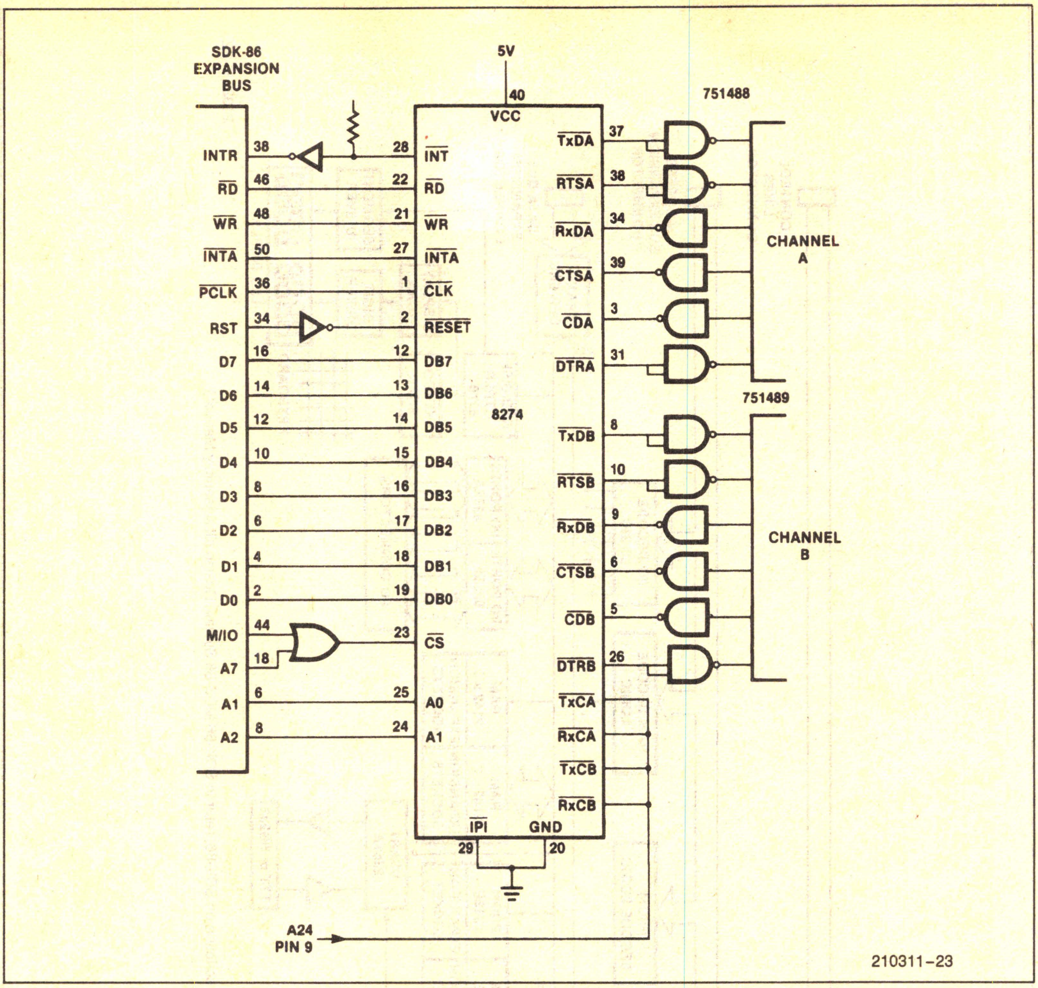


Figure D-1. 8274/SDK-86 Hardware Interface

The channel operating modes are contained in two tables starting with line #163. As the 8274 has only one command register per channel, the remaining seven registers are loaded indirectly through the WR0 (Write Register 0) register. The first byte of each table entry is the register pointer value which is loaded into WR0 and the second byte is the value for that particular register.

The indicated modes set the 8274 for asynchronous operation with data characters 8 bits long, no parity, and 2 stop bits. An X16 baud rate clock is assumed. Also selected is the "interrupt on all RX character" mode with a variable interrupt vector compatible with the 8086/8088. The transmitters are enabled and all model control lines are put in their active state.

In addition to initializing the 8274, this routine also sets up the appropriate interrupt vectors. The 8086 assumes the first 1K bytes of memory contain up to 256 separate interrupt vectors. On the SDK-86 the initial 2K bytes of memory is RAM and therefore must be initialized with the appropriate vectors. (In a prototype system, this initial memory is probably ROM, thus the vector set-up is not needed.) The 8274 supplies up to eight different interrupt vectors. These vectors are developed from internal conditions such as data requests, status changes, or error conditions for each channel. The initialization routine arbitrarily assumes that the initial 8274 vector corresponds to 8086 vector location 80H (memory location 200H). This choice is arbitrary since the 8274 initial vector location is programmable.

Finally, the initialization routine sets up the status and flag in RAM. The meaning and use of these locations are discussed later.

Following the initialization routine are those for the transmit commands (starting with line #268). These commands assume that the host CPU has initialized the publicly declared variables for the transmit buffer pointer, TX_POINTER_CHx, and the buffer length, TX_LENGTH_CHx. The transmit command routines simply clear the transmitter empty flag, TX EMPTY CHx, and load the first character of the buffer into the transmitter. It is necessary to load the first character in this manner since transmitter interrupts are generated only when the 8274's transmit data buffer becomes empty. It is the act of becoming empty which generates the interrupt not simply the buffer being empty, thus the transmitter needs one character to start.

The host CPU can monitor the transmitter empty flag, TX_EMPTY_CHx, in order to determine when transmission of the buffer is complete. Obviously, the CPU should only call the command routine after first checking that the empty flag is set.

After returning to the main program, all transmitter data transfers are handled via the transmitter-interrupt service routines starting at lines #360 and #443. These routines start by issuing an End-Of-Interrupt command to the 8274. (This command resets the internal-interrupt controller logic of the 8274 for this particular vector and opens the logic for other internal interrupt requests. The routines next check the length count. If the buffer is completely transmitted, the transmitter empty flag, TX_EMPTY_CHx, is set and a command is issued to the 8274 to reset its interrupt line. Assuming that the buffer is not completely transmitted, the next character is output to the transmitter. In either case, an interrupt return is executed to return to the main CPU program.

The receiver commands start at line #314. Like the transmit commands, it is assumed that the CPU has initialized the receive-buffer-pointer public variable, RX_POINTER_CHx. This variable points to the first location in an empty receive buffer. The command routines clear the receiver ready flag, RX_READY_CHx, and then set the receiver enable bit in the 8274 WR3 register. With the receiver now enabled, any received characters are placed in the receive buffer using interrupt-driven data transfers.

The received data service routines, starting at lines #402 and #485, simply place the received character in the buffer after first issuing the EOI command. The character is then compared to an ASCII CR. An ASCII CR causes the routine to set the receiver ready flag, RX_READY_CHx, and to disable the receiver. The CPU can interrogate this flag to determine when the buffer contains a new line of data. The receive buffer pointer, RX_POINTER_CHx, points to the last received character and the receive counter, RX_COUNTER_CHx, contains the length.

That completes our discussion of the command routines and their associated interrupt service routines. Although not used by the commands, two additional service routines are included for completeness. These routines handle the error and status-change interrupt vectors.

The error service routines, starting at lines #427 and #510, are vectored to if a special receive condition is detected by the 8274. These special receive conditions include parity, receiver overrun, and framing errors. When this vector is generated, the error condition is indicated in RR1 (Read Register 1). The error service routine issues an EOI command, reads RR1 and places it in the ERROR_MSG_CHx variable, and then issues a reset error command to the 8274. The CPU can monitor the error message location to detect error conditions. The designer, of course, can supply his own error service routine.

Similarly, the status-change routines (starting lines #386 and #469) are initiated by a change in the modem-control status lines CTS/, CD/, or SYNDET/. (Note that WR2 bit 0 controls whether the 8274 generates interrupts based upon changes in these lines. Our WR2 parameter is such that the 8274 is programmed to ignore changes for these inputs.) The service routines simply read RR0, place its contents in the STATUS_MSG_CHx variable and then issue a reset external status command. Read Register 0 contains the state of the modem inputs at the point of the last change.

Well, that's it. This application example has presented useful, albeit very simple, routines showing how the 8274 might be used to transmit and receive buffers using an asynchronous serial format. Extensions for byte- or bit-synchronous formats would require no hardware changes due to the highly programmable nature of the 8274's serial formats.

8274 APPLICATION BRIEF PROGRAM

```
ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ASYNCB
OBJECT MODULE PLACED IN :F1:ASYNCB.OBJ
ASSEMBLER INVOKED BY:  ASM86 :F1:ASYNCB.SRC

LOC  OBJ                 LINE     SOURCE

                         ;****************************************************************
                         ;*                                                              *
                         ;*                  8274 APPLICATION BRIEF PROGRAM              *
                         ;*                                                              *
                         ;*                                                              *
                         ;*                                                              *
                         ;* THE 8274 IS INITIALIZED FOR SIMPLE ASYNCHRONOUS SERIAL       *
                         ;* FORMAT AND VECTORED INTERRUPT-DRIVEN DATA TRANSFERS.         *
                         ;* THE INITIALIZATION ROUTINE ALSO LOADS THE 8086'S INTERRUPT   *
                         ;* VECTOR TABLE FROM THE CODE SEGMENT INTO LOW RAM ON THE       *
                         ;* SDK-86.  THE TRANSMITTER AND RECEIVER ARE LEFT ENABLED.      *
                         ;*                                                              *
                         ;* FOR TRANSMIT, THE CPU PASSES IN MEMORY THE POINTER OF A      *
                         ;* BUFFER TO TRANSMIT AND THE BYTE LENGTH OF THE BUFFER.        *
                         ;* THE DATA TRANSFER PROCEED USING INTERRUPT-DRIVEN TRANSFERS.  *
                         ;* A STATUS BIT IN MEMORY IS SET WHEN IF BUFFERS IS EMPTY.      *
                         ;*                                                              *
                         ;* FOR RECEIVE, THE CPU PASSES THE POINTER OF A BUFFER TO FILL. *
                         ;* THE BUFFER IS FILLED UNTIL A 'CR_CHR' CHARACTER IS RECEIVED. *
                         ;* A STATUS BIT IS SET AND THE CPU MAY READ THE RX POINTER TO   *
                         ;* DETERMINE THE LOCATION OF THE LAST CHARACTER.                *
                         ;*                                                              *
                         ;* ALL ROUTINES ARE ASSUMED TO EXIST IN THE SAME CODE SEGMENT.  *
                         ;* CALL'S TO THE SERVICE ROUTINES ARE ASSUMED TO BE "SHORT" OR  *
                         ;* INTRASEGMENT (ONLY THE RETURN ADDRESS IP IS ON THE STACK).   *
                         ;*                                                              *
                         ;*                                                              *
                         ;*                                                              *
                         ;*                                                              *
                         ;****************************************************************
```

210311-24

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ          LINE    SOURCE


                                NAME    ASYNCB  ;MODULE NAME

                          ;PUBLIC DECLARATIONS FOR COMMAND ROUTINES

                                PUBLIC  INITIAL_8274    ;INITIALIZATION ROUTINE
                                PUBLIC  TX_COMMAND_CHB  ;TX BUFFER COMMAND CHANNEL B
                                PUBLIC  TX_COMMAND_CHA  ;TX BUFFER COMMAND CHANNEL A
                                PUBLIC  RX_COMMAND_CHB  ;RX BUFFER COMMAND CHANNEL B
                                PUBLIC  RX_COMMAND_CHA  ;RX BUFFER COMMAND CHANNEL A

                          ;PUBLIC DECLARATIONS FOR STATUS VARIABLES

                                PUBLIC  RX_READY_CHB    ;RX READY FLAG CHB
                                PUBLIC  RX_READY_CHA    ;RX READY FLAG CHA
                                PUBLIC  TX_EMPTY_CHB    ;TX EMPTY FLAG CHB
                                PUBLIC  TX_EMPTY_CHA    ;TX EMPTY FLAG CHA
                                PUBLIC  RX_COUNT_CHB    ;RX BUFFER COUNTER CHB
                                PUBLIC  RX_COUNT_CHA    ;RX BUFFER COUNTER CHA
                                PUBLIC  ERROR_MSG_CHB   ;ERROR FLAG CHB
                                PUBLIC  ERROR_MSG_CHA   ;ERROR FLAG CHA
                                PUBLIC  STATUS_MSG_CHB  ;STATUS FLAG CHB
                                PUBLIC  STATUS_MSG_CHA  ;STATUS FLAG CHA

                          ;PUBLIC DECLARATIONS FOR VARIABLES PASSED TO THE TRANSMIT
                          ;AND RECEIVE COMMANDS.

                                PUBLIC  TX_POINTER_CHB  ;TX BUFFER POINTER FOR CHB
                                PUBLIC  TX_LENGTH_CHB   ;TX LENGTH OF BUFFER FOR CHB
                                PUBLIC  TX_POINTER_CHA  ;TX BUFFER POINTER FOR CHA
                                PUBLIC  TX_LENGTH_CHA   ;TX LENGTH OF BUFFER FOR CHA
                                PUBLIC  RX_POINTER_CHB  ;RX BUFFER POINTER FOR CHB
                                PUBLIC  RX_POINTER_CHA  ;RX BUFFER POINTER FOR CHA

                          ;I/O PORT ASSIGNMENTS

                          ;CHANNEL A PORT ASSIGNMENTS

  0000                    DATA_PORT_CHA         EQU     0                  ;DATA I/O PORT
  0002                    COMMAND_PORT_CHA      EQU     2                  ;COMMAND PORT
  0002                    STATUS_PORT_CHA       EQU     COMMAND_PORT_CHA   ;STATUS PORT

                          ;CHANNEL B PORT ASSIGNMENTS

  0004                    DATA_PORT_CHB         EQU     4                  ;DATA I/O PORT
  0006                    COMMAND_PORT_CHB      EQU     6                  ;COMMAND PORT
  0006                    STATUS_PORT_CHB       EQU     COMMAND_PORT_CHB   ;STATUS PORT

                          ;MISC. SYSTEM EQUATES

  000D                    CR_CHR          EQU   0DH     ;ASCII CR CHARACTER CODE
  0200                    INT_TABLE_BASE  EQU   200H    ;INT. VECTOR BASE ADDRESS
  0500                    CODE_START      EQU   500H    ;START LOCATION FOR CODE

                          $EJECT

                          ;RAM ASSIGNMENTS FOR DATA SEGMENT

----                            DATA    SEGMENT

```

210311-25

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ          LINE    SOURCE

                          ;VECTOR INTERRUPT TABLE - ASSUME INITIAL 8274 INTERRUPT
                          ;VECTOR IS NUMBER 80 (0200H)    FOR EACH VECTOR, THE TABLE
                          ;CONTAINS START LOCATION AND CODE SEGMENT REGISTER VALUE
                          ;THE TABLE IS LOADED FROM PROM

0200                              ORG     INT_TABLE_BASE

0200 0000                 TX_VECTOR_CHB   DW      0       ;TX INTERRUPT VECTOR FOR CHB
0202 0000                 TX_CS_CHB       DW      0

0204 0000                 STS_VECTOR_CHB  DW      0       ;STATUS INTERRUPT VECTOR FOR CHB
0206 0000                 STS_CS_CHB      DW      0

0208 0000                 RX_VECTOR_CHB   DW      0       ;RX INTERRUPT VECTOR FOR CHB
020A 0000                 RX_CS_CHB       DW      0

020C 0000                 ERR_VECTOR_CHB  DW      0       ;ERROR INTERRUPT VECTOR FOR CHB
020E 0000                 ERR_CS_CHB      DW      0

0210 0000                 TX_VECTOR_CHA   DW      0       ;TX INTERRUPT VECTOR FOR CHA
0212 0000                 TX_CS_CHA       DW      0

0214 0000                 STS_VECTOR_CHA  DW      0       ;STATUS INTERRUPT VECTOR FOR CHA
0216 0000                 STS_CS_CHA      DW      0

0218 0000                 RX_VECTOR_CHA   DW      0       ;RX INTERRUPT VECTOR FOR CHA
021A 0000                 RX_CS_CHA       DW      0

021C 0000                 ERR_VECTOR_CHA  DW      0       ;ERROR INTERRUPT VECTOR FOR CHA
021E 0000                 ERR_CS_CHA      DW      0

                          ;MISC RAM LOCATIONS FOR CHANNEL STATUS AND POINTERS

                          ;CHANNEL B POINTERS AND STATUS

0220 0000                 TX_POINTER_CHB  DW      0       ;TX BUFFER POINTER FOR CHB
0222 0000                 TX_LENGTH_CHB   DW      0       ;TX BUFFER LENGTH FOR CHB
0224 0000                 RX_POINTER_CHB  DW      0       ;RX BUFFER POINTER FOR CHB
0226 0000                 RX_COUNT_CHB    DW      0       ;RX LENGTH COUNTER FOR CHB
0228 00                   TX_EMPTY_CHB    DB      0       ;TX DONE FLAG
0229 00                   RX_READY_CHB    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
022A 00                   STATUS_MSG_CHB  DB      0       ;STATUS CHANGE MESSAGE
022B 00                   ERROR_MSG_CHB   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)

                          ;CHANNEL A POINTERS AND STATUS

022C 0000                 TX_POINTER_CHA  DW      0       ;TX BUFFER POINTER FOR CHA
022E 0000                 TX_LENGTH_CHA   DW      0       ;TX BUFFER LENGTH FOR CHA
0230 0000                 RX_POINTER_CHA  DW      0       ;RX BUFFER POINTER FOR CHA
0232 0000                 RX_COUNT_CHA    DW      0       ;RX LENGTH COUNTER FOR CHA
0234 00                   TX_EMPTY_CHA    DB      0       ;TX DONE FLAG
0235 00                   RX_READY_CHA    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
0236 00                   STATUS_MSG_CHA  DB      0       ;STATUS CHANGE MESSAGE
0237 00                   ERROR_MSG_CHA   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)

----                              DATA    ENDS

                     +1   $EJECT
```

210311-26

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ            SOURCE

----                        ABC     SEGMENT
                            ASSUME  CS:ABC,DS:DATA,SS:DATA
0500                        ORG     CODE_START

                    ;*****************************************************************
                    ;*                                                               *
                    ;*            PARAMETERS FOR CHANNEL INITIALIZATION              *
                    ;*                                                               *
                    ;*****************************************************************

                    ;CHANNEL B PARAMETERS

                            ;WR1 - INTERRUPT ON ALL RX CHR, VARIABLE INT VECTOR, TX INT ENABLE
0500 01             CMDSTRB DB      1,16H
0501 16
                            ;WR2 - INTERRUPT VECTOR
0502 02                     DB      2,(INT_TABLE_BASE/4)
0503 80
                            ;WR3 - RX 8 BITS/CHR, RX DISABLE
0504 03                     DB      3,0C0H
0505 C0
                            ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0506 04                     DB      4,4CH
0507 4C
                            ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0508 05                     DB      5,0EAH
0509 EA
                            ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
050A 00                     DB      0,0
050B 00

                    ;CHANNEL A PARAMETERS

                            ;WR1 - INTERRUPT ON ALL RX CHR, TX INT ENABLE
050C 01             CMDSTRA DB      1,12H
050D 12
                            ;WR2 - VECTORED INTERRUPT FOR 8086
050E 02                     DB      2,30H
050F 30
                            ;WR3 - RX 8 BITS/CHR, RX DISABLE
0510 03                     DB      3,0C0H
0511 C0
                            ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0512 04                     DB      4,4CH
0513 4C
                            ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0514 05                     DB      5,0EAH
0515 EA
                            ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
0516 00                     DB      0,0
0517 00

                    $EJECT
```

210311–27

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ              LINE    SOURCE


                              ;START OF COMMAND ROUTINES

                              ;***************************************************************
                              ;*                                                             *
                              ;*      INITIALIZATION COMMAND FOR THE 8274 - THE 8274         *
                              ;*      IS SETUP ACCORDING TO THE PARAMETERS STORED IN         *
                              ;*      PROM ABOVE STARTING AT CMSTRB FOR CHANNEL B AND        *
                              ;*      CMSTRA FOR CHANNEL A.                                  *
                              ;*                                                             *
                              ;***************************************************************

0518                          INITIAL_8274:
                                      ;COPY INTERRUPT VECTOR IP AND CS VALUES FROM PROM TO RAM
0518 C70600020806                     MOV     TX_VECTOR_CHB, OFFSET XMTINB    ;TX DATA VECTOR CHB
051E 8C0E0202                         MOV     TX_CS_CHB, CS
0522 C70604023506                     MOV     STS_VECTOR_CHB, OFFSET STAINB   ;STATUS VECTOR CHB
0528 8C0E0602                         MOV     STS_CS_CHB, CS
052C C70608024906                     MOV     RX_VECTOR_CHB, OFFSET RCVINB    ;RX DATA VECTOR CHB
0532 8C0E0A02                         MOV     RX_CS_CHB, CS
0536 C7060C027706                     MOV     ERR_VECTOR_CHB, OFFSET ERRINB   ;ERROR VECTOR CHB
053C 8C0E0A02                         MOV     RX_CS_CHB, CS
0540 C70610028C06                     MOV     TX_VECTOR_CHA, OFFSET XMTINA    ;TX DATA VECTOR CHA
0546 8C0E1202                         MOV     TX_CS_CHA, CS
054A C7061402B906                     MOV     STS_VECTOR_CHA, OFFSET STAINA   ;STATUS VECTOR CHA
0550 8C0E1602                         MOV     STS_CS_CHA, CS
0554 C7061802CD06                     MOV     RX_VECTOR_CHA, OFFSET RCVINA    ;RX DATA VECTOR CHA
055A 8C0E1A02                         MOV     RX_CS_CHA, CS
055E C7061C02FB06                     MOV     ERR_VECTOR_CHA, OFFSET ERRINA   ;ERROR VECTOR CHA
0564 8C0E1E02                         MOV     ERR_CS_CHA, CS

                              ;COPY SETUP TABLE PARAMETERS INTO 8274

0568 BF0005                           MOV     DI, OFFSET CMDSTRB      ;INITIALIZE CHB
056B BA0600                           MOV     DX, COMMAND_PORT_CHB
056E E82E00                           CALL    SETUP                   ;COPY CHB PARAMETERS
0571 BF0C05                           MOV     DI, OFFSET CMDSTRA      ;INITIALIZE CHA
0574 BA0200                           MOV     DX, COMMAND_PORT_CHA
0577 E82500                           CALL    SETUP                   ;COPY CHA PARAMETERS

                              ;INITIALIZE STATUS BYTES AND FLAGS

057A B80000                           MOV     AX, 0
057D A22B02                           MOV     ERROR_MSG_CHB, AL       ;CLEAR ERROR FLAG CHB
0580 A23702                           MOV     ERROR_MSG_CHA, AL       ;CLEAR ERROR FLAG CHA
0583 A22A02                           MOV     STATUS_MSG_CHB, AL      ;CLEAR STATUS FLAG CHB
0586 A23602                           MOV     STATUS_MSG_CHA, AL      ;CLEAR STATUS FLAG CHA
0589 A32602                           MOV     RX_COUNT_CHB, AX        ;CLEAR RX COUNTER CHB
058C A33202                           MOV     RX_COUNT_CHA, AX        ;CLEAR RX COUNTER CHA
058F B001                             MOV     AL, 1
0591 A22902                           MOV     RX_READY_CHB, AL        ;SET RX DONE FLAG CHB
0594 A23502                           MOV     RX_READY_CHA, AL        ;SET RX DONE FLAG CHA
0597 A22802                           MOV     TX_EMPTY_CHB, AL        ;SET TX DONE FLAG CHB
059A A23402                           MOV     TX_EMPTY_CHA, AL        ;SET TX DONE FLAG CHA
059D FB                               STI                             ;ENABLE INTERRUPTS
059E C3                               RET                             ;RETURN - DONE WITH SETUP

059F 8A05                     SETUP:  MOV     AL, [DI]                ;PARAMETER COPYING ROUTINE
05A1 3C00                             CMP     AL, 0
05A3 7404                             JE      DONE
```

210311-28

```
LOC  OBJ                  LINE     SOURCE

05A5 EE                                    OUT    DX, AL                ;OUTPUT PARAMETER
05A6 47                                    INC    DI                    ;POINT AT NEXT PARAMETER
05A7 EBF6                                  JMP    SETUP                 ;GO LOAD IT
05A9 C3                            DONE:   RET                          ;DONE - SO RETURN

                                +1 $EJECT

                                   ;*****************************************************************
                                   ;*                                                               *
                                   ;*    TX CHANNEL B COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                                   ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                                   ;*    TX_POINTER_CHB, AND THE BUFFER LENGTH, TX_LENGTH_CHB,      *
                                   ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.                *
                                   ;*    BOTH ITEMS ARE WORD VARIABLES.                             *
                                   ;*                                                               *
                                   ;*****************************************************************

05AA                               TX_COMMAND_CHB:
05AA 50                                    PUSH   AX               ;SAVE REGISTERS
05AB 57                                    PUSH   DI
05AC 52                                    PUSH   DX
05AD C606280200                            MOV    TX_EMPTY_CHB, 0  ;CLEAR EMPTY FLAG
05B2 BA0400                                MOV    DX, DATA_PORT_CHB        ;SETUP PORT POINTER
05B5 8B3E2002                              MOV    DI, TX_POINTER_CHB       ;GET TX BUFFER POINTER CHB
05B9 8A05                                  MOV    AL, [DI]         ;GET FIRST CHARACTER TO TX
05BB EE                                    OUT    DX, AL           ;OUTPUT IT TO 8274 TO GET IT STARTED
05BC 5A                                    POP    DX
05BD 5F                                    POP    DI
05BE 58                                    POP    AX
05BF C3                                    RET                     ;RETURN

                                   ;*****************************************************************
                                   ;*                                                               *
                                   ;*    TX CHANNEL A COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                                   ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                                   ;*    TX_POINTER_CHA, AND THE BUFFER LENGTH, TX_LENGTH_CHA,      *
                                   ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.                *
                                   ;*    BOTH ITEMS ARE WORD VARIABLES.                             *
                                   ;*                                                               *
                                   ;*****************************************************************

05C0                               TX_COMMAND_CHA:
05C0 50                                    PUSH   AX               ;SAVE REGISTERS
05C1 57                                    PUSH   DI
05C2 52                                    PUSH   DX
05C3 C606340200                            MOV    TX_EMPTY_CHA, 0  ;CLEAR EMPTY FLAG
05C8 BA0000                                MOV    DX, DATA_PORT_CHA        ;SETUP PORT POINTER
05CB 8B3E2C02                              MOV    DI, TX_POINTER_CHA       ;GET TX BUFFER POINTER CHA
05CF 8A05                                  MOV    AL, [DI]         ;GET FIRST CHARACTER TO TX
05D1 EE                                    OUT    DX, AL           ;OUTPUT IT TO 8274 TO GET IT STARTED
05D2 5A                                    POP    DX
05D3 5F                                    POP    DI
05D4 58                                    POP    AX
05D5 C3                                    RET                     ;RETURN

                                   ;*****************************************************************
                                   ;*                                                               *
                                   ;*    RX COMMAND FOR CHANNEL B - THE CALLING ROUTINE MUST        *
                                   ;*    INITIALIZE RX_POINTER_CHB TO POINT AT THE RECEIVE          *
                                   ;*    BUFFER BEFORE CALLING THIS ROUTINE.                        *
```

210311-29

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ              SOURCE

                      ;*                                                            *
                      ;**************************************************************

05D6                  RX_COMMAND_CHB:
05D6 50                       PUSH    AX                      ;SAVE REGISTERS
05D7 52                       PUSH    DX
05D8 C606290200               MOV     RX_READY_CHB, 0 ;CLEAR RX READY FLAG
05DD C70626020000             MOV     RX_COUNT_CHB, 0 ;CLEAR RX COUNTER
05E3 BA0600                   MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
05E6 B003                     MOV     AL, 3                   ;SET UP FOR WR3
05E8 EE                       OUT     DX, AL
05E9 B0C1                     MOV     AL, 0C1H                ;WR3 - 8 BITS/CHR, ENABLE RX
05EB EE                       OUT     DX, AL
05EC 5A                       POP     DX
05ED 58                       POP     AX
05EE C3                       RET                             ;RETURN

                      ;**************************************************************
                      ;*                                                            *
                      ;*     RX COMMAND FOR CHANNEL A - THE CALLING ROUTINE MUST    *
                      ;*     INITIALIZE RX_POINTER_CHA TO POINT AT THE RECEIVE      *
                      ;*     BUFFER BEFORE CALLING THIS ROUTINE                     *
                      ;*                                                            *
                      ;**************************************************************

05EF                  RX_COMMAND_CHA:
05EF 50                       PUSH    AX                      ;SAVE REGISTERS
05F0 52                       PUSH    DX
05F1 C606350200               MOV     RX_READY_CHA, 0 ;CLEAR RX READY FLAG
05F6 C70632020000             MOV     RX_COUNT_CHA, 0 ;CLEAR RX COUNTER
05FC BA0200                   MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
05FF B003                     MOV     AL, 3                   ;SET UP FOR WR3
0601 EE                       OUT     DX, AL
0602 B0C1                     MOV     AL, 0C1H                ;WR3 - 8 BITS/CHR, ENABLE RX
0604 EE                       OUT     DX, AL
0605 5A                       POP     DX
0606 58                       POP     AX
0607 C3                       RET                             ;RETURN

                   +1 $EJECT

                      ;**************************************************************
                      ;*                                                            *
                      ;*           START OF INTERRUPT SERVICE ROUTINES              *
                      ;*                                                            *
                      ;**************************************************************

                      ;CHANNEL B TRANSMIT DATA SERVICE ROUTINE

0608 52               XMTINB: PUSH    DX                      ;SAVE REGISTERS
0609 57                       PUSH    DI
060A 50                       PUSH    AX
060B E80201                   CALL    EOI                     ;SEND EOI COMMAND TO 8274
060E FF062002                 INC     TX_POINTER_CHB  ;POINT TO NEXT CHARACTER
0612 FF0E2202                 DEC     TX_LENGTH_CHB   ;DEC LENGTH COUNTER
0616 740E                     JE      XIB             ;TEST IF DONE
0618 BA0400                   MOV     DX, DATA_PORT_CHB       ;NOT DONE - GET NEXT CHARACTER
061B 8B3E2002                 MOV     DI, TX_POINTER_CHB
061F 8A05                     MOV     AL, [DI]                ;PUT CHARACTER IN AL
0621 EE                       OUT     DX, AL                  ;OUTPUT IT TO 8274
```

210311-30

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ           SOURCE

0622 58                     POP    AX              ;RESTORE REGISTERS
0623 5F                     POP    DI
0624 5A                     POP    DX
0625 CF                     IRET                   ;RETURN TO FOREGROUND
0626 BA0600         XIB:    MOV    DX, COMMAND_PORT_CHB    ;ALL CHARACTERS HAVE BEEN SEND
0629 B028                   MOV    AL, 28H         ;RESET TRANSMITTER INTERRUPT PENDING
062B EE                     OUT    DX, AL
062C C606280201             MOV    TX_EMPTY_CHB, 1 ;DONE - SO SET TX EMPTY FLAG CHB
0631 58                     POP    AX              ;RESTORE REGISTERS
0632 5F                     POP    DI
0633 5A                     POP    DX
0634 CF                     IRET                   ;RETURN TO FOREGROUND
                    ;CHANNEL B STATUS CHANGE SERVICE ROUTINE
0635 52             STAINB: PUSH   DX              ;SAVE REGISTERS
0636 57                     PUSH   DI
0637 50                     PUSH   AX
0638 E8D500                 CALL   EOI             ;SEND EOI COMMAND TO 8274
063B BA0600                 MOV    DX, COMMAND_PORT_CHB
063E EC                     IN     AL, DX          ;READ RR0
063F A22A02                 MOV    STATUS_MSG_CHB, AL      ;PUT RR0 IN STATUS MESSAGE
0642 B010                   MOV    AL, 10H         ;SEND RESET STATUS INT COMMAND TO 8274
0644 EE                     OUT    DX, AL
0645 58                     POP    AX              ;RESTORE REGISTERS
0646 5F                     POP    DI
0647 5A                     POP    DX
0648 CF                     IRET
                    ;CHANNEL B RECEIVED DATA SERVICE ROUTINE
0649 52             RCVINB: PUSH   DX              ;SAVE REGISTERS
064A 57                     PUSH   DI
064B 50                     PUSH   AX
064C E8C100                 CALL   EOI             ;SEND EOI COMMAND TO 8274
064F 8B3E2402               MOV    DI, RX_POINTER_CHB      ;GET RX CHB BUFFER POINTER
0653 BA0400                 MOV    DX, DATA_PORT_CHB
0656 EC                     IN     AL, DX          ;READ CHARACTER
0657 8805                   MOV    [DI], AL        ;STORE IN BUFFER
0659 FF062402               INC    RX_POINTER_CHB  ;BUMP THE BUFFER POINTER
065D FF062602               INC    RX_COUNT_CHB    ;BUMP THE COUNTER
0661 3C0D                   CMP    AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
0663 750E                   JNE    RIB
0665 C606290201             MOV    RX_READY_CHB, 1 ;YES, SET READY FLAG
066A BA0600                 MOV    DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
066D B003                   MOV    AL, 3           ;POINT AT WR3
066F EE                     OUT    DX, AL
0670 B0C0                   MOV    AL, 0C0H        ;DISABLE RX
0672 EE                     OUT    DX, AL
0673 58             RIB:    POP    AX              ;EITHER WAY, RESTORE REGISTERS
0674 5F                     POP    DI
0675 5A                     POP    DX
0676 CF                     IRET                   ;RETURN TO FOREGROUND
                    ;CHANNEL B ERROR SERVICE ROUTINE
0677 52             ERRINB: PUSH   DX              ;SAVE REGISTERS
0678 50                     PUSH   AX
0679 E89400                 CALL   EOI             ;SEND EOI COMMAND TO 8274
067C BA0600                 MOV    DX, COMMAND_PORT_CHB
```

210311-31

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ            LINE    SOURCE

067F B001                           MOV     AL, 1               ;POINT AT RR1
0681 EE                             OUT     DX, AL
0682 EC                             IN      AL, DX              ;READ RR1
0683 A22B02                         MOV     ERROR_MSG_CHB, AL         ;SAVE IT IN ERROR FLAG
0686 B030                           MOV     AL, 30H             ;SEND RESET ERROR COMMAND TO 8274
0688 EE                             OUT     DX, AL
0689 58                             POP     AX                  ;RESTORE REGISTERS
068A 5A                             POP     DX
068B CF                             IRET                        ;RETURN TO FOREGROUND

                            ;CHANNEL A TRANSMIT DATA SERVICE ROUTINE

068C 52                     XMTINA: PUSH    DX                  ;SAVE REGISTERS
068D 57                             PUSH    DI
068E 50                             PUSH    AX
068F E87E00                         CALL    EOI                 ;SEND EOI COMMAND TO 8274
0692 FF062C02                       INC     TX_POINTER_CHA  ;POINT TO NEXT CHARACTER
0696 FF0E2E02                       DEC     TX_LENGTH_CHA   ;DEC LENGTH COUNTER
069A 740E                           JE      XIA                 ;TEST IF DONE
069C BA0000                         MOV     DX, DATA_PORT_CHA         ;NOT DONE - GET NEXT CHARACTER
069F 8B3E2C02                       MOV     DI, TX_POINTER_CHA
06A3 8A05                           MOV     AL, [DI]            ;PUT CHARACTER IN AL
06A5 EE                             OUT     DX, AL              ;OUTPUT IT TO 8274
06A6 58                             POP     AX                  ;RESTORE REGISTERS
06A7 5F                             POP     DI
06A8 5A                             POP     DX
06A9 CF                             IRET                        ;RETURN TO FOREGROUND
06AA BA0200                 XIA:    MOV     DX, COMMAND_PORT_CHA      ;ALL CHARACTERS HAVE BEEN SEND
06AD B028                           MOV     AL, 28H             ;RESET TRANSMITTER INTERRUPT PENDING
06AF EE                             OUT     DX, AL
06B0 C606340201                     MOV     TX_EMPTY_CHA, 1 ;DONE - SO SET TX EMPTY FLAG CHB
06B5 58                             POP     AX                  ;RESTORE REGISTERS
06B6 5F                             POP     DI
06B7 5A                             POP     DX
06B8 CF                             IRET                        ;RETURN TO FOREGROUND

                            ;CHANNEL A STATUS CHANGE SERVICE ROUTINE

06B9 52                     STAINA: PUSH    DX                  ;SAVE REGISTERS
06BA 57                             PUSH    DI
06BB 50                             PUSH    AX
06BC E85100                         CALL    EOI                 ;SEND EOI COMMAND TO 8274
06BF BA0200                         MOV     DX, COMMAND_PORT_CHA
06C2 EC                             IN      AL, DX              ;READ RR0
06C3 A23602                         MOV     STATUS_MSG_CHA, AL        ;PUT RR0 IN STATUS MESSAGE
06C6 B010                           MOV     AL, 10H             ;SEND RESET STATUS INT COMMAND TO 8274
06C8 EE                             OUT     DX, AL
06C9 58                             POP     AX                  ;RESTORE REGISTERS
06CA 5F                             POP     DI
06CB 5A                             POP     DX
06CC CF                             IRET

                            ;CHANNEL A RECEIVED DATA SERVICE ROUTINE

06CD 52                     RCVINA: PUSH    DX                  ;SAVE REGISTERS
06CE 57                             PUSH    DI
06CF 50                             PUSH    AX
06D0 E83D00                         CALL    EOI                 ;SEND EOI COMMAND TO 8274
06D3 8B3E3002                       MOV     DI, RX_POINTER_CHA        ;GET RX CHA BUFFER POINTER
06D7 BA0000                         MOV     DX, DATA_PORT_CHA
```

210311-32

```
MCS-86 MACRO ASSEMBLER    ASYNCB


LOC  OBJ                 LINE     SOURCE

06DA EC                                  IN      AL, DX            ;READ CHARACTER
06DB 8805                                MOV     [DI], AL          ;STORE IN BUFFER
06DD FF063002                            INC     RX_POINTER_CHA    ;BUMP THE BUFFER POINTER
06E1 FF063202                            INC     RX_COUNT_CHA      ;BUMP THE COUNTER
06E5 3C0D                                CMP     AL, CR_CHR        ;TEST IF LAST CHARACTER TO BE RECEIVED?
06E7 750E                                JNE     RIA
06E9 C606350201                          MOV     RX_READY_CHA, 1 ;YES, SET READY FLAG
06EE BA0200                              MOV     DX, COMMAND_PORT_CHA      ;POINT AT COMMAND PORT
06F1 B003                                MOV     AL, 3             ;POINT AT WR3
06F3 EE                                  OUT     DX, AL
06F4 B0C0                                MOV     AL, 0C0H          ;DISABLE RX
06F6 EE                                  OUT     DX, AL
06F7 58                          RIA:    POP     AX                ;EITHER WAY, RESTORE REGISTERS
06F8 5F                                  POP     DI
06F9 5A                                  POP     DX
06FA CF                                  IRET                      ;RETURN TO FOREGROUND

                                 ;CHANNEL A ERROR SERVICE ROUTINE

06FB 52                          ERRINA: PUSH    DX                ;SAVE REGISTERS
06FC 50                                  PUSH    AX
06FD E81000                              CALL    EOI               ;SEND EOI COMMAND TO 8274
0700 BA0200                              MOV     DX, COMMAND_PORT_CHA
0703 B001                                MOV     AL, 1             ;POINT AT RR1
0705 EE                                  OUT     DX, AL
0706 EC                                  IN      AL, DX            ;READ RR1
0707 A23702                              MOV     ERROR_MSG_CHA, AL         ;SAVE IT IN ERROR FLAG
070A B030                                MOV     AL, 30H           ;SEND RESET ERROR COMMAND TO 8274
070C EE                                  OUT     DX, AL
070D 58                                  POP     AX                ;RESTORE REGISTERS
070E 5A                                  POP     DX
070F CF                                  IRET                      ;RETURN TO FOREGROUND

                                  END-OF-INTERRUPT ROUTINE - SENDS EOI COMMAND TO 8274.
                                    THIS COMMAND MUST ALWAYS TO ISSUED ON CHANNEL A.

0710 50                          EOI:    PUSH    AX                ;SAVE REGISTERS
0711 52                                  PUSH    DX
0712 BA0200                              MOV     DX, COMMAND_PORT_CHA      ;ALWAYS FOR CHANNEL A !!!
0715 B038                                MOV     AL, 38H
0717 EE                                  OUT     DX, AL
0718 5A                                  POP     DX
0719 58                                  POP     AX
071A C3                                  RET

                                 ;END OF CODE ROUTINE

----                                     ABC     ENDS
                                         END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

210311-33

REFERENCES

1. 8274 Multiprotocol Serial Controller (MPSC) Data Sheet, Intel Corporation, California, 1980.
2. Basics of Data Communication, Electronics Book Series, McGraw-Hill, New York, 1976.
3. Telecommunications and the Computer, J. Martin, Prentice-Hall, New Jersey, 1976.
4. Technical Aspects of Data Communications, J. McNamara, DEC Press, Massachusetts, 1977.
5. Miscellaneous Data Communications Standards—EIA RS-232-C, EIA RS-422, EIA RS-423, EIA Standard Sales, Washington, D.C.

intel®

APPLICATION NOTE

AP-145

November 1986

Synchronous Communication with the 8274 Multiple Protocol Serial Controller

SIKANDAR NAQVI
APPLICATION ENGINEER

Order Number: 210403-001

INTRODUCTION

The INTEL 8274 is a Multi-Protocol Serial Controller, capable of handling both asynchronous and synchronous communication protocols. Its programmable features allow it to be configured in various operating modes, providing opimization to given data communication application.

This application note describes the features of the MPSC in Synchronous Communication applications only. It is strongly recommended that the reader read the 8274 Data Sheet and Application Note AP134 "Asynchronous Communication with the 8274 Multi-Protocol Serial Controller" before reading this Application Note. This Application note assumes that the reader is familiar with the basic structure of the MPSC, in terms of pin description, Read/Write registers and asynchronous communication with the 8274. Appendix A contains the software listings of the Application Example and Appendix B shows the MPSC Read/Write Registers for quick reference.

The first section of this application note presents an overview of the various synchronous protocols. The second section discusses the block diagram description of the MPSC. This is followed by the description of MPSC interrupt structure and mode of operation in the third and fourth sections. The fifth section describes a hardware/software example, using the INTEL single board computer iSBC88/45 as the hardware vehicle. The sixth section consists of some specialized applications of the MPSC. Finally, in section seven, some useful programming hints are summarized.

SYNCHRONOUS PROTOCOL OVERVIEW

This section presents an overview of various synchronous protocols. The contents of this section are fairly tutorial and may be skipped by the more knowledgeable reader.

Bit Oriented Protocols Overview

Bit oriented protocols have been defined to manage the flow of information on data communication links. One of the most widely known protocols is the one defined by the International Standards Organization: HDLC (High Level Data Link Control). The American Standards Association's protocol, ADCCP is similar to HDLC. CCITT Recommendation X.25 layer 2 is also an acceptable version of HDLC. Finally, IBM's SDLC (Synchronous Data Link Control) is also a subset of the HDLC.

In this section, we will concentrate most of our discussion on HDLC. Figure 1 shows a basic HDLC frame format.

A frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide, extendable to 2 or more bytes. The control field is also 8 bits wide, extendable to two bytes. The data field or information field may be any number of bits. The data field may or may not be on an 8-bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags.

ZERO BIT INSERTION

The flag has a unique binary bit pattern: 7E HEX. To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8274 performs zero bit insertion and deletion automatically in the SDLC/HDLC mode. The zero-bit stuffing ensures periodic transitions in the data stream. These transitions are necessary for a phase lock circuit, which may be used at the receiver end to generate a receive clock which is in phase to the received data. The inserted and deleted 0's are not included in the CRC checking. The *address* field is used to address a given secondary station. The *control* field contains the link-level control information which includes implied acknowledgement, supervisory commands and responses, etc. A more detailed discussion of higher level protocol functions is beyond the scope of this application note. Interested readers may refer to the references at the end of this application note.

Opening Flag Byte	Address* Field (A)	Control** Field (C)	Data Field	Frame Check Sequence	Closing Flag Byte

Figure 1. HDLC/SDLC Frame Format

*Extendable to 2 or More Bytes.
**Extendable to 2 Bytes.

The *data field* may be of any length and content in HDLC. Note that SDLC specifies that data field be a multiple of bytes only. In data communications, it is generally desirable to transmit data which may be of any content. This requires that data field should not contain characters which are defined to assist the transmission protocol (like opening flag 7EH in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion discussed earlier and the bit oriented nature of the protocol.

The last field is the FCS (Frame Check Sequence). The FCS uses the error detecting techniques called Cyclic Redundancy Check. In SDLC/HDLC, the CCITT-CRC must be used.

NON-RETURN TO ZERO INVERTED (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC protocol. It allows HDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for a phase lock circuit at the receiver end to derive a receive clock (from received data) which is synchronized to the received data and at the same time ensure data transparency.

Byte Synchronous Communication

As the name implies, Byte Synchronous Communication is a synchronous communication protocol which means that the transmitting station is synchronized to the receiving station through the recognition of a special sync character or characters. Two examples of Byte Synchronous protocol are the IBM Bisync and Monosync. Bisync has two starting sync characters per message while monosync has only one sync character. For the sake of brevity, we will only discuss Bisync here. All the discussion is valid for Monosync also. Any exceptions will be noted. Figure 2 shows a typical Bisync message format.

The Bisync protocol is defined for half duplex communication between two or more stations over point to point or multipoint communication lines. Special characters control link access, transmission of data and termination of transmission operations for the system. A detailed discussion of these special control characters (SYN, ENQ, STX, ITB, ETB, ETX, DLE, SOH, ACK0, ACK1, WACK, NAK and EOT, etc) is beyond the scope of this Application Note. Readers interested in more detailed discussion are directed to the references listed at the end of this Application Note.

As shown in Figure 2, each message is preceded by two sync characters. Since the sync characters are defined at the beginning of the message only, the transmitter must insert fill characters (sync) in order to maintain synchronization with the receiver when no data is being transmitted.

TRANSPARENT TRANSMISSION

Bisync protocol requires special control characters to maintain the communication link over the line. If the data is EBCDIC encoded, then transparency is ensured by the fact that the field will not contain any of the bisync control characters. However, if data does not conform to standard character encoding techniques, transparency in bisync is achieved by inserting a special character DLE (Data Link Escape) before and after a string of characters which are to be transmitted transparently. This ensures that any data characters which match any of the special characters are not confused for special characters. An example of a transparent block is shown in Figure 3.

In a transparent mode, it is required that the CRC (BCC) is not performed on special characters. Later on, we will show how the 8274 can be used to achieve transparent transmission in Bisync mode.

SYNC	SYNC	SOH	HEADER	STX TEXT	ETX OR ETB	CRC 1	CRC 2

Figure 2. Bisync Message Format

DLE	STX	TRANSPARENT TRANSMISSION	DLE	ETX	BCC

Enter transparent mode — return to normal mode

Figure 3. Bisync Transparent Format

BLOCK DIAGRAM

This section discusses the block diagram view of the 8274. The CPU interface and serial interface is discussed separately. This will be followed by a hardware example in the fifth section, which will show how to interface the 8274 with the Intel CPU 8088. The 8274 block diagram is shown in Figure 4.

CPU Interface

The CPU interface to the system interface logic block utilizes the A0, A1, $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$ inputs to communicate with the internal registers of the 8274. Figure 5 shows the address of the internal registers. The DMA interface is achieved by utilizing DMA request lines for each channel: $TxDRQ_A$, $TxDRQ_B$, $RxDRQ_A$, $RxDRQ_B$. Note that $TxDRQ_B$ and $RxDRQ_B$ become $\overline{IPO}$ and $\overline{IPI}$ respectively in non-DMA mode. $\overline{IPI}$ is the Interrupt Priority Input and $\overline{IPO}$ is the Interrupt Priority Output. These two pins can be used for connecting multiple MPSCs in a daisy chain. If the Wait Mode is programmed, then $TxRDQ_A$ and $RxDRQ_A$ pins become RDY_B and RDY_A pins. These pins can be wire-OR'ed and are usually hooked up to the CPU RDY line to synchronize the CPU for block transfers. The $\overline{INT}$ pin is activated whenever the MPSC requires CPU attention. The $\overline{INTA}$ may be used to utilize the powerful vectored mode feature of the 8274. Detailed discussion on these subjects will be done later in this Application Note. The RESET pin may be used for hardware reset while the clock is required to click the internal logic on the MPSC.

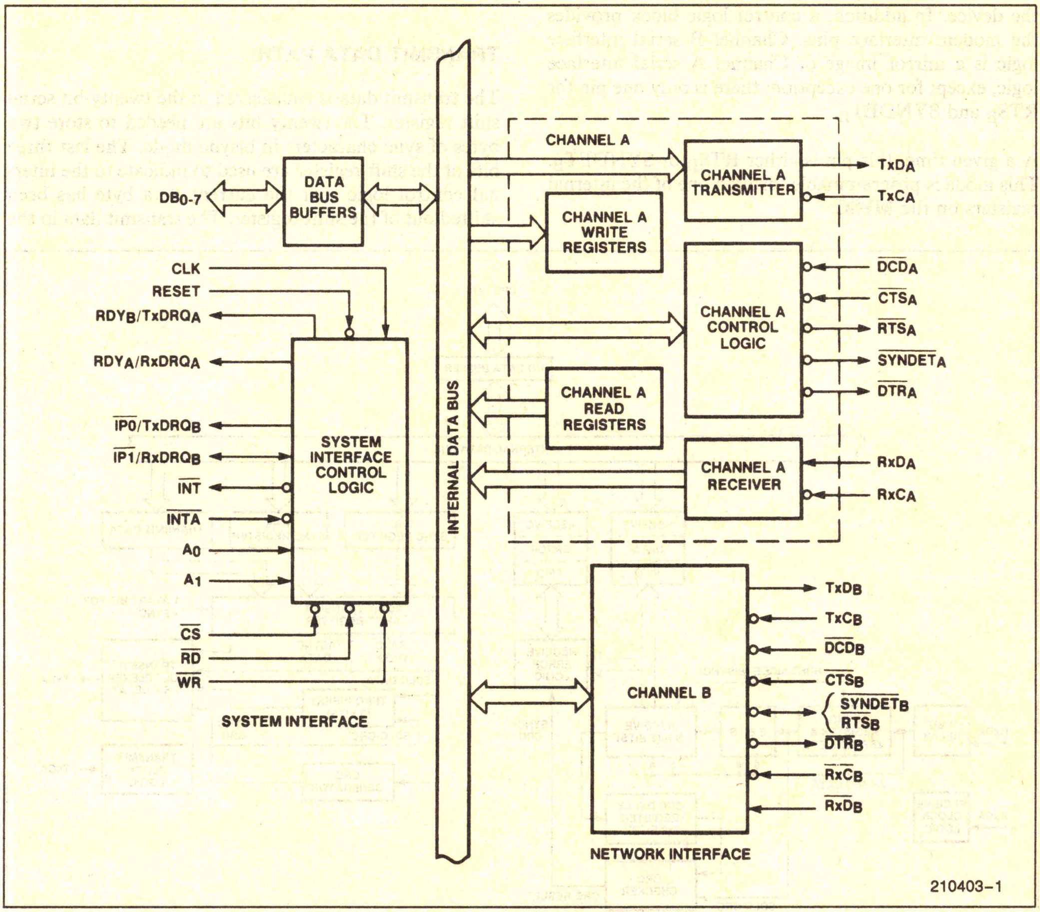


Figure 4. 8274 Block Diagram

$\overline{CS}$	A1	A0	Read Operation	Write Operation
0	0	0	CHA DATA READ	CHA DATA WRITE
0	1	0	CHA STATUS REGISTER (RR0,RR1)	CHA COMMAND/PARAMETER (WR0–WR7)
0	0	1	CHB DATA READ	CHB DATA WRITE
0	1	1	CHB STATUS REGISTER (RR0,RR1,RR2)	CHB COMMAND/PARAMETER (WR0–WR7)
1	X	X	HIGH Z	HIGH Z

Figure 5. Bus Interface

Serial Interface

On the serial side, there are two completely independent channels: Channel A and Channel B. Each channel consists of a transmitter block, receiver block and a set of read/write registers which are used to initialize the device. In addition, a control logic block provides the modem interface pins. Channel B serial interface logic is a mirror image of Channel A serial interface logic, except for one exception: there is only one pin for RTS_B and $SYNDET_B$.

A a given time, this pin is either RTS_B or $SYNDET_B$. This mode is programmable through one of the internal registers on the MPSC.

Transmit and Receive Data Path

Figure 6 shows a block diagram for transmit and receive data path. Without describing each block on the diagram, a brief discussion of the block diagram will be presented here.

TRANSMIT DATA PATH

The transmit data is transferred to the twenty-bit serial shift register. The twenty bits are needed to store two bytes of sync characters in bisync mode. The last three bits of the shift register are used to indicate to the internal control logic that the current data byte has been shifted out of the shift register. The transmit data in the

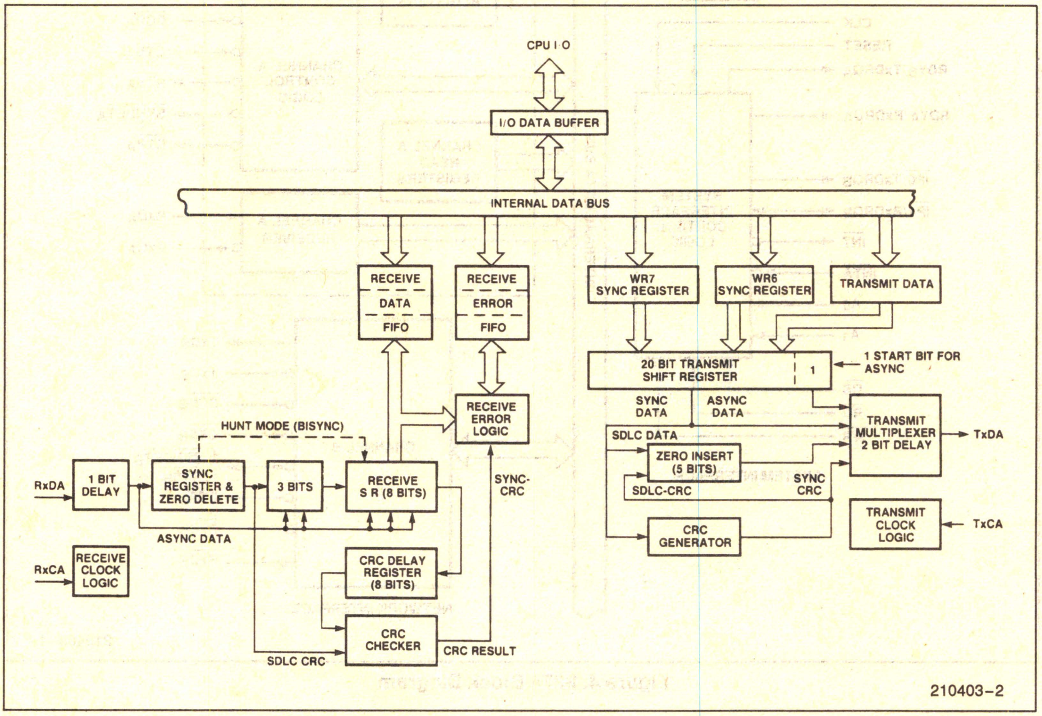


Figure 6. Transmit and Receive Data Path

transmit shift register is shifted out through a two bit delay onto the TxData line. This two bit delay is used to synchronize the internal shift clock with the external transmit clock. The data in the shift register is also presented to zero bit insertion logic which inserts a zero after sensing five contiguous ones in the data stream. In parallel to all this activity, the CRC-generator is computing CRC on the transmitted data and appends the frame with CRC bytes at the end of the data transmission.

RECEIVE DATA PATH

The received data is passed through a one bit delay before it is presented for flag/sync comparison. In bisync mode, after the synchronization is achieved, the incoming data bypasses the sync register and enters directly into the three bit buffer on its way to receive shift register. In SDLC mode, the incoming data always passes through the sync register where the data pattern is continuously monitored for contiguous ones for the

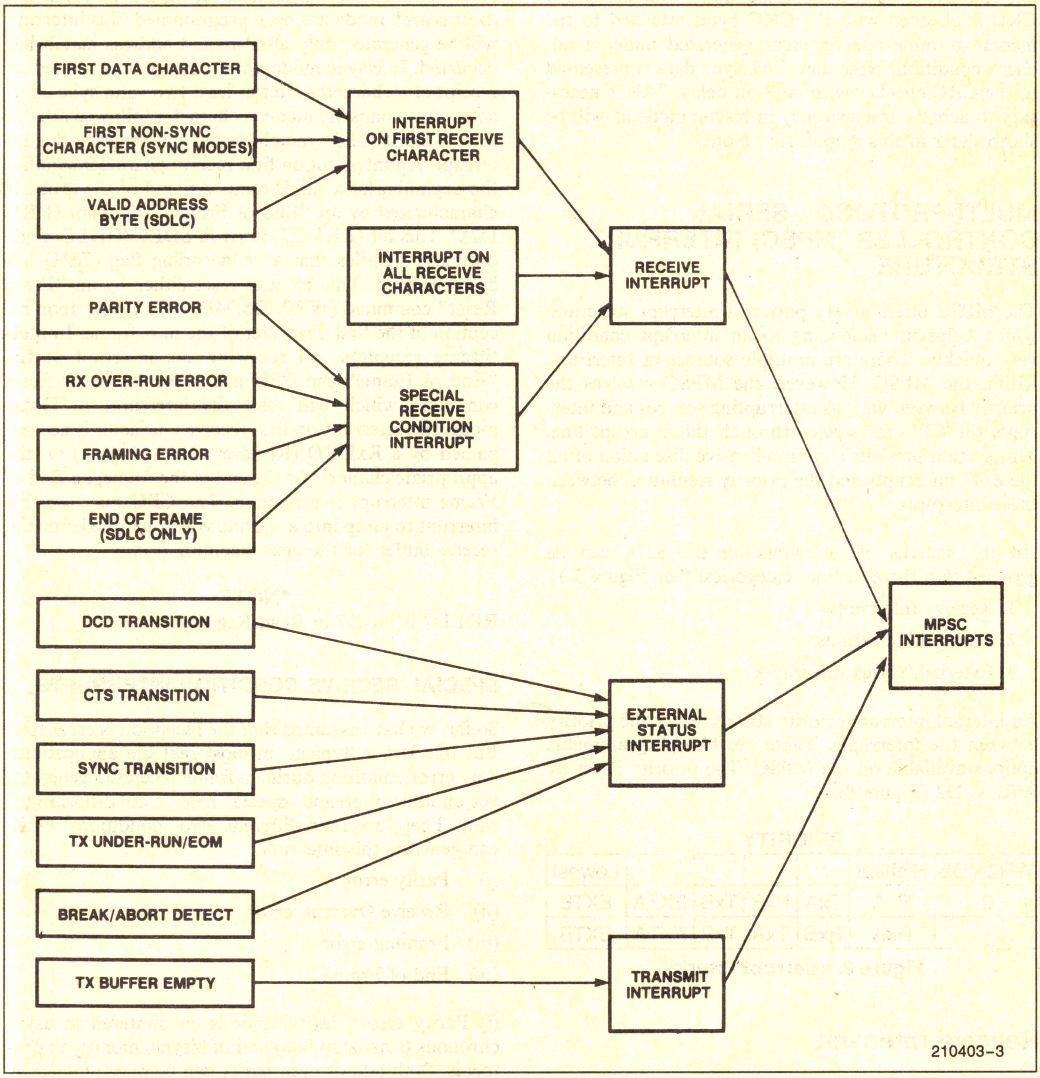


Figure 7. MPSC Interrupt Structure

zero deletion logic. The data then enters the three bit buffer and the receive shift register. From the receive shift register, the data is transferred to the three byte deep FIFO. The data is transferred to the top of the FIFO at the chip clock rate (not the receiver clock). It takes three chip clock/periods to transfer data from the serial shift register to the top of the FIFO. The three bit deep Receive Error FIFO shifts any error condition which may have occurred during a frame reception. While all this is happening, the CRC checker is checking the CRC on the incoming data. The computed CRC is checked with the CRC bytes attached to the incoming frame and an error generated under a no-check condition. Note that the bisync data is presented to the CRC checker with an 8-bit delay. This is necessary to achieve transparency in bisync mode as will be shown later in this Application Note.

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) INTERRUPT STRUCTURE

The MPSC offers a very powerful interrupt structure, which helps in responding to an interrupt condition very quickly. There are multiple sources of interrupts within the MPSC. However, the MPSC resolves the priority between various interrupting sources and interrupts the CPU for service through the interrupt line. This section presents a comprehensive discussion of all the 8247 interrupts and the priority resolution between these interrupts.

All the sources of interrupts on the 8274 can be grouped into three distinct categories. (See Figure 7.)

1. Receive Interrupts
2. Transmit Interrupts
3. External/Status Interrupts.

An internal interrupt priority structure sets the priority between the interrupts. There are two programmable options available on the MPSC. The priority is set by WR2A, D2 (Figure 8).

PRIORITY

WR2A:D2	Highest					Lowest
0	RxA	TxA	RxB	TxB	EXTA	EXTB
1	RxA	RxB	TxA	TxB	EXTA	EXTB

Figure 8. Interrupt Priority

Receive Interrupt

All receive interrupts may be categorized into two distinct groups: Receive Interrupt on Receive Character and Special Receive Condition Interrupts.

RECEIVE INTERRUPT ON RECEIVE CHARACTER

A receive interrupt is generated when a character is received by the MPSC. However, as will be discussed later, this is a programmable feature on the MPSC. A Rx character available interrupt is generated by the MPSC after the receive character has been assembled by the MPSC. It may be noted that in DMA transfer mode too, a receive interrupt on the first receive character should be programmed. In SDLC mode, if address search mode has been programmed, this interrupt will be generated only after a valid address match has occurred. In bisync mode, this interrupt is generated on receipt of a character after at least two valid sync characters. In monosync mode, a character followed after at least a single valid sync character will generate this interrupt. An interrupt on first receive character signifies the beginning of a valid frame. An end of the frame is characterized by an "End of Frame" Interrupt (RR1: D7).* This bit (RR1:D7) is set in SDLC/HDLC mode only and signifies that a valid ending flag (7EH) has been received. This bit gets reset either by an "Error Reset" command (WR0: D5D4D3 = 110) or upon reception of the first character of the next frame. In multiframe reception, on receiving the interrupt at the "End of Frame" the CPU may issue an Error Reset command which will reset the interrupt. In DMA mode, the interrupt on first receive character is accompanied by a RxDRQ (Receiver DMA request) on the appropriate channel. At the end of the frame, an End of Frame interrupt is generated. The CPU may use this interrupt to jump into a routine which may redefine the receive buffer for the next incoming frame.

*NOTE:
RR1:D7 is bit D7 in Read Register 1.

SPECIAL RECEIVE CONDITION INTERRUPTS

So far, we have assumed that the reception is error free. But this is not 'typical' in most real life applications. Any error condition during a frame reception generates yet another interrupt—special receive condition interrupt. There are four different error conditions which can generate this interrupt.

(i) Parity error
(ii) Receive Overrun error
(iii) Framing error
(iv) End of Frame

(i) Parity error: Parity error is encountered in asynchronous (start-stop bits) and in bisync/monosync protocols. Both odd or even parity can be programmed. A parity error in a received byte will generate a special receive condition interrupt and sets bit 4 in RR1.

(ii) Receive Overrun error: If the CPU or the DMA controller (in DMA mode) fails to read a received character within three byte times after the received character interrupt (or DMA request) was generated, the receiver buffer will overflow and this will generate a special receive condition interrupt and sets bit 5 in RR1.

(iii) Framing error: In asynchronous mode, a framing error will generate a special receive interrupt and set bit D6 in RR1. This bit is not latched and is updated on the next received character.

(iv) End of frame: This interrupt is encountered in SDLC/HDLC mode only. When the MPSC receives the closing flag, it generates the special receive condition interrupt and sets bit D7 in RR1.

All the special receive condition interrupts may be reset by issuing an Error Reset Command.

CRC Error: In SDLC/HDLC and synchronous modes, a CRC error is indicated by bit D6 in RR1. When used to check CRC error, this bit is normally set until a correct CRC match is obtained which resets this bit. After receiving a frame, the CPU must read this bit (RR1:D6) to determine if a valid CRC check had occurred. It may be noted that a CRC error does not generate an interrupt.

It may also be pointed out that in SDLC/HDLC mode, receive DMA requests are disabled by a special receive condition and can only be re-enabled by issuing an Error Reset Command.

Transmit Interrupt

A transmit buffer empty generates a transmit interrupt. This has been discussed earlier under "Transmit in Interrupt Mode" and it would be sufficient to note here that a transmit buffer empty interrupt is generated only when the transmit buffer gets empty—assuming it had a data character loaded into it earlier. This is why on starting a frame transmission, the first data character is loaded by the CPU without a transmit empty interrupt (or DMA request in DMA mode). After this character is loaded into the serial shift register, the buffer becomes empty, and an interrupt (or DMA request) is generated. This interrupt is reset by a "Reset Tx Interrupt/DMA Pending" command (WR0: D5 D4 D3 = 101).

External/Status Interrupt

Continuing our discussion on transmit interrupt, if the transmit buffer is empty and the transmit serial shift register also becomes empty (due to the data character shifted out of the MPSC), a transmit under-run interrupt will be generated. This interrupt may be reset by "Reset External/Status Interrupt" command (WR0: D5 D4 D3 = 101).

The External Status Interrupt can be caused by five different conditions:

(i) CD Transition

(ii) CTS Transition

(iii) Sync/Hunt Transition

(iv) Tx under-run/EOM condition

(v) Break/Abort Detection.

CD, CTS TRANSITION

Any transition on these inputs on the serial interface will generate an External/Status interrupt and set the corresponding bits in status register RR0. This interrupt will also be generated in DMA as well as in Wait Mode. In order to find out the state of the $\overline{CTS}$ or $\overline{CD}$ pins *before* the transition had occurred, RR0 must be read before issuing a Reset External/Status Command through WR0. A read of RR0 after the Reset External/Status Command will give the condition of $\overline{CTS}$ or $\overline{CD}$ pins *after* the transition had occurred. Note that bit D5 in RR0 gives the complement of the state of $\overline{CTS}$ pin while D3 in RR0 reflects the actual state of the $\overline{CD}$ pin.

SYNC HUNT TRANSITION

Any transition of the $\overline{SYNDET}$ input generates an interrupt. However, sync input has different functions in different modes and we shall discuss them individually.

SDLC Mode

In SDLC mode, the $\overline{SYNDET}$ pin is an output. Status register RR1, D4 contains the state of the $\overline{SYNDET}$ pin. The Enter Hunt Mode initially sets this bit in R0. An opening flag in a received SDLC frame resets this bit and generates an external status interrupt. Every time the receiver is enabled or the Enter Hunt Code Command is issued, an external status interrupt will be generated on receiving a valid flag followed by a valid address/data character. This interrupt may be reset by the "Reset External/Status Interrupt" command.

External SYNC Mode

The MPSC can be programmed into External Sync Mode by setting WR4, D5 D4 = 11. The $\overline{SYNDET}$ pin is an input in this case and must be held high until an external character synchronization is established. However, the External Sync mode is enabled by the Enter Hunt Mode control bit (WR3: D4). A high at the $\overline{SYNDET}$ pin holds the Sync/Hunt bit (RR0,D4) in the reset state. When external synchronization is established, $\overline{SYNDET}$ must be driven low on second rising

edge of RxC after the rising edge of RxC on which the last bit of sync character was received. This high to low transition sets the Sync/Hunt bit and generates an external/status interrupt, which must be reset by the Reset External/Status command. If the $\overline{\text{SYNDET}}$ input goes high again, another External Status Interrupt is generated, which may be cleared by Reset External/ Status command.

Mono-Sync/Bisync Mode

$\overline{\text{SYNDET}}$ pin acts as an output in this case. The Enter Hunt Mode sets the Sync/Hunt bit in R0. Sync/Hunt bit is reset when the MPSC achieves character synchronization. This high to low transition will generate an external status interrupt. The $\overline{\text{SYNDET}}$ pin goes active every time a sync pattern is detected in the data stream. Once again, the external status interrupt may be reset by the Reset External/Status command.

Tx UNDER-RUN/END OF MESSAGE (EOM)

The transmitter logic includes a transmit buffer and a transmit serial shift register. The CPU loads the character into the transmit buffer which is transferred into the transmit shift register to be shifted out of the MPSC. If the transmit buffer gets empty, a transmit buffer empty interrupt is generated (as discussed earlier). However, if the transmit buffer gets empty *and* the serial shift register gets empty, a transmit under-run condition will be created. This generates an External Status Interrupt and the interrupt can be cleared by the Reset External Status command. The status register RR0, D6 bit is set when the transmitter under-runs. This bit plays an important role in controlling a transmit operation, as will be discussed later in this application note.

BREAK/ABORT DETECTION

In asynchronous mode, bit D7 in RR0 is set when a break condition is detected on the receive data line. This also generates an External/Status interrupt which may be reset by issuing a Reset External/Status Interrupt command to the MPSC. Bit D7 in RR0 is reset when the break condition is terminated on the receive data line and this causes another External/Status interrupt to ge generated. Again, a Reset External/Status Interrupt command will reset this interrupt and will enable the break detection logic to look for the next break sequence.

In SDLC Receive Mode, an Abort sequence (seven or more 1's) detection on the receive data line will generate an External/Status interrupt and set RR0,D7. A Reset External/Status command will clear this interrupt. However, a termination of the Abort sequence will generate another interrupt and set RR0,D7 again. Once again, it may be cleared by issuing Reset External/Status Command.

This concludes our discussion on External Status Interrupts.

Interrupt Priority Resolution

The internal interrupt priority between various interrupt sources is resolved by an internal priority logic circuit, according to the priority set in WR2A. We will now discuss the interrupt timings during the priority resolution. Figures 9 and 10 show the timing diagrams for vectored and non-vectored modes.

VECTORED MODE

We shall assume that the MPSC accepted an internal request for an interrupt by activating the internal INT signal. This leads to generating an external interrupt signal on the $\overline{\text{INT}}$ pin. The CPU responds with an interrupt acknowledge (INTA) sequence. The leading edge of the first $\overline{\text{INTA}}$ pulse sets an internal interrupt acknowledge signal (we will call it Internal INTA). Internal INTA is reset by the high going edge of the third $\overline{\text{INTA}}$ pulse. The MPSC will not accept any internal requests for an interrupt during the period when Internal $\overline{\text{INTA}}$ is active (high). The MPSC resolves the priority during various existing internal interrupt requests during the Interrupt Request Priority Resolve Time, which is defined as the time between the leading edge of the first INTA and the leading edge of the second INTA from the CPU. Once the internal priorities have been resolved, an internal Interrupt-in-service Latch is set. The external $\overline{\text{INT}}$ is also deactivated when the Interrupt-in-Service Latch is set.

The lower priority interrupt requests are not accepted internally until an EOI (WR0: D5 D4 D3 = 111 Ch. A only) command is issued by the CPU. The EOI command enables the lower priority interrupts. However, a higher priority interrupt request will still be accepted (except during the period when internal $\overline{\text{INTA}}$ is active) even though the Internal-in-Service Latch is set.

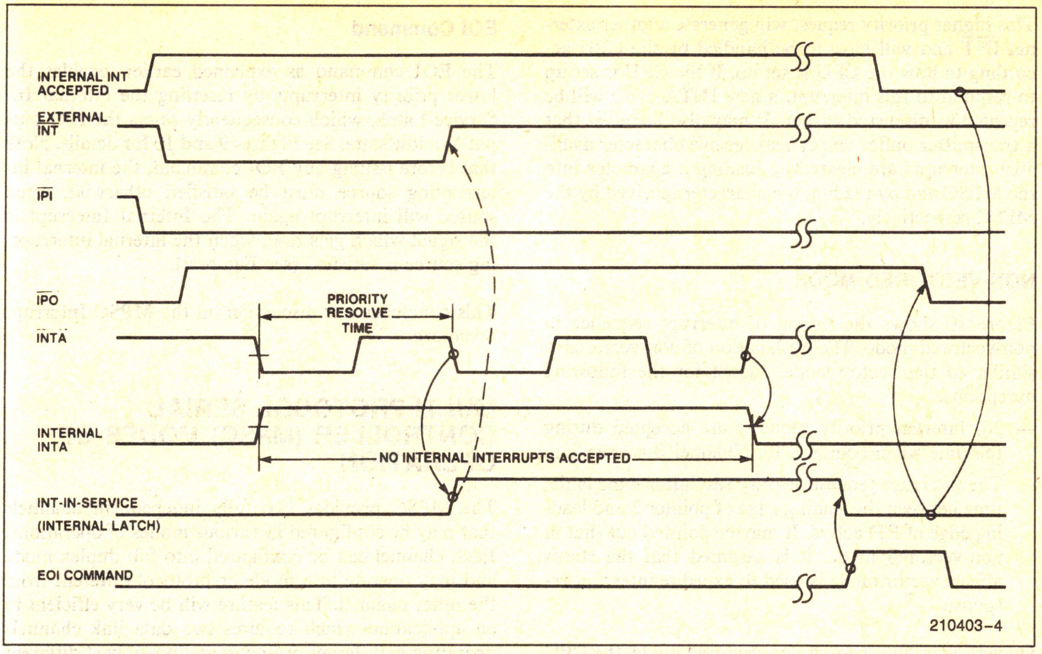


Figure 9. 8274 in 8085 Vectored Mode Priority Resolution Time

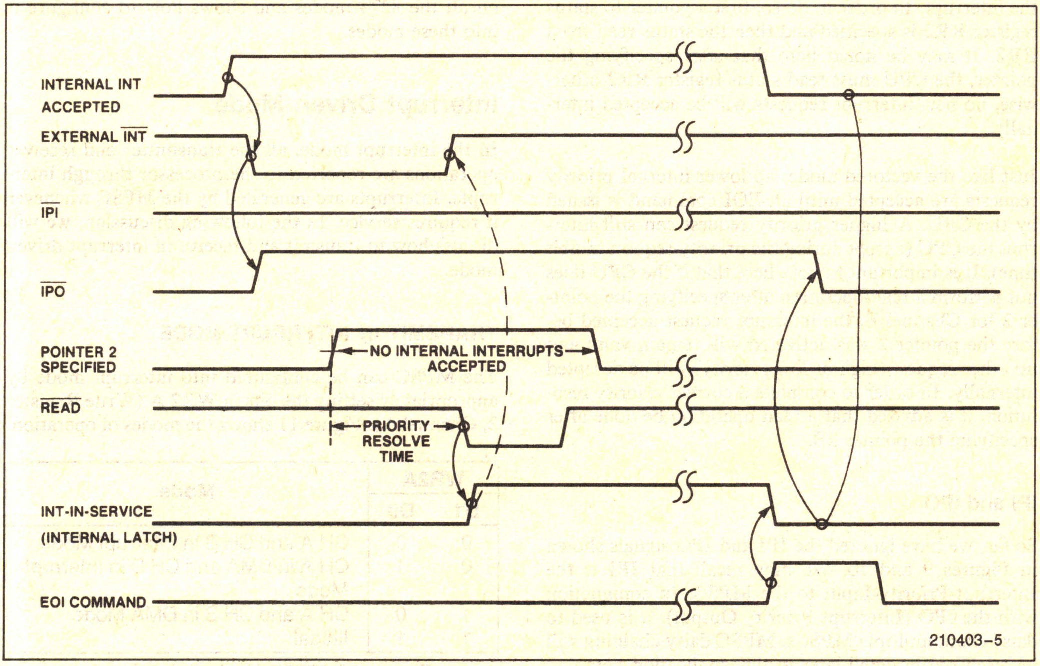


Figure 10. 8274 Non Vectored Mode Priority Resolve Time

This higher priority request will generate another external $\overline{INT}$ and will have to be handled by the CPU according to how the CPU is set up. If the CPU is set up to respond to this interrupt, a new INTA cycle will be repeated as discussed earlier. It may also be noted that a transmitter buffer empty and receive character available interrupts are cleared by loading a character into the MPSC and by reading the character received by the MPSC respectively.

NON-VECTORED MODE

Figure 10 shows the timing of interrupt sequence in non-vectored mode. The explanation of non-vectored is similar to the vector mode, except for the following exceptions.

— No internal priority requests are accepted during the time when pointer 2 for Channel B is specified.

— The interrupt request priority resolution time is the time between the leading edge of pointer 2 and leading edge of RD active. It may be pointed out that in non-vectored mode, it is assumed that the status affects vector mode is used to expedite interrupt response.

On getting an interrupt in non-vectored mode, the CPU must read status register RR2 to find out the cause of the interrupt. In order to do so, first a pointer to status register RR2 is specified and then the status read from RR2. It may be noted here that after specifying the pointer, the CPU must read status register RR2 otherwise, no new interrupt requests will be accepted internally.

Just like the vectored mode, no lower internal priority requests are accepted until an EOI command is issued by the CPU. A higher priority request can still interrupt the CPU (except during the priority request inhibit time). It is important to note here that if the CPU does not perform a read operation after specifying the pointer 2 for Channel B, the interrupt request accepted before the pointer 2 was activated will remain valid and no other request (high or low priority) will be accepted internally. In order to complete a correct priority resolution, it is advised that a read operation be done after specifying the pointer 2B.

IPI and IPO

So far, we have ignored the IPI and IPO signals shown in Figures 9 and 10. We may recall that IPI is the Interrupt-Priority-Input to the MPSC. In conjunction with the IPO (Interrupt Priority Output), it is used to daisy chain multiple MPSCs. MPSC daisy chaining will be discussed in detail later in this application note.

EOI Command

The EOI command as explained earlier, enables the lower priority interrupts by resetting the internal In-Service-Latch, which consequently resets the IPO output to a low state. See Figures 9 and 10 for details. Note that before issuing any EOI command, the internal interrupting source must be satisfied otherwise, same source will interrupt again. The Internal Interrupt is the signal which gets reset when the internal interrupting source is satisfied (see Figure 9).

This concludes our discussion on the MPSC Interrupt Structure.

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) MODES OF OPERATION

The MPSC provides two fully independent channels that may be configured in various modes of operations. Each channel can be configured into full duplex mode and may operate in a mode or protocol different from the other channel. This feature will be very efficient in an application which requires two data link channels operating in different protocols and possibly at different data rates. This section presents a detailed discussion on all the 8274 modes and shows how to configure it into these modes.

Interrupt Driven Mode

In the interrupt mode, all the transmitter and receiver operations are reported to the processor through interrupts. Interrupts are generated by the MPSC whenever it requires service. In the following discussion, we will discuss how to transmit and receive in interrupt driven mode.

TRANSMIT IN INTERRUPT MODE

The MPSC can be configured into interrupt mode by appropriately setting the bits in WR2 A (Write Register 2, Channel A). Figure 11 shows the modes of operation.

WR2A		Mode
D1	D0	
0	0	CH A and CH B in Interrupt Mode
0	1	CH A in DMA and CH B in Interrupt Mode
1	0	CH A and CH B in DMA Mode
1	1	Illegal

Figure 11. MPSC Mode Selection for Channel A and Channel B

We will limit our discussion to SDLC transmit and receive only. However, exceptions for other synchronous protocols will be pointed out. To initiate a frame transmission, the first data character must be loaded from the CPU, in all cases. (DMA Mode too, as you will notice later in this application note). Note that in SDLC mode, this first data character may be the address of the station addressed by the MPSC. The transmit buffer consists of a transmit buffer and a serial shift register. When the character is transferred from the buffer into the serial shift regiser, an interrupt due to transmit buffer empty is generated. The CPU has one byte time to service this interrupt and load another character into the transmitter buffer. The MPSC will generate an interrupt due to transmit buffer underrun condition if the CPU does not service the Transmit Buffer Empty Interrupt within one byte time.

This process will continue until the CPU is out of any more data characters to be sent. At this point, the CPU does not respond to the interrupt with a character but simply issues a Reset Tx INT/DMA pending command (WR0: D5 D4 D3 = 101). The MPSC will ultimately underrun, which simply means that both the transmit buffer and transmit shift registers are empty. At this point, flag character (7EH) or CRC byte is loaded into the transmit shift register. This sets the transmit underrun bit in RR0 and generates "Transmit Underrun/EOM" interrupt (RR0: D6 = 1).

You will recall that an SDLC frame has two CRC bytes after the data field. 8274 generates the CRC on all the data that is loaded from the CPU. During initialization, there is a choice of selecting a CRC-16 or CCITT-CRC (WR5: D2). In SDLC/HDLC operation, CCITT-CRC must be selected. We will now see how the CRC gets inserted at the end of the data field. Here we have a choice of having the CRC attached to the data field or sending the frame without the CRC bytes. During transmission, a "Reset Tx Underrun/EOM Latch" command (WR0: D7 D6 = 11) will ensure that at the end of the frame when the transmitter underruns, CRC bytes will be automatically inserted at the end of the data field. If the "Reset Tx Underrun/EOM Latch" command was not issued during the transmission of data characters, no CRC would be inserted and the MPSC will transmit flags (7EH) instead.

However, in case of CRC transmission, the CRC transmission sets the Tx Underrun/EOM bit and generates a Transmitter Underrun/EOM Interrupt as discussed earlier. This will have to be reset in the next frame to ensure CRC insertion in the next frame. It is recommended that Tx Underrun/EOM latch be reset very early in the transmission mode, preferably after loading the first character. It may be noted here that Tx Underrun EOM latch cannot be reset if there is no data in the transmit buffer. This means that at least one character has to be loaded into the MPSC before a "Reset Transmit Underrun/EOM Latch" command will be accepted by the MPSC.

When the transmitter is underrun, an interrupt is generated. This interrupt is generated at the beginning of the CRC transmission, thus giving the user enough time (minimum 22 transmit clock cycles) to issue an Abort command (WR0: D5 D4 D3 = 0 0 1) in case if the transmitted data had an error. The Abort Command will ensure that the MPSC transmits at least eight 1's but less than fourteen 1's before the line reverts to continuous flags. The receiver will scratch this frame because of bad CRC.

However, assuming the transmission was good (no Abort Command issued), after the CRC bytes have been transmitted, closing flag (7EH) is loaded into the transmit buffer. When the flag (7EH) byte is transferred to the serial shift register, a transmit buffer empty interrupt is generated. If another frame has to be transmitted, a new data character has to be loaded into the transmit buffer and the complete transmit sequence repeated. If no more frames are to be transmitted, a "Reset Transmit INT/DMA Pending" command (WR0: D5 D4 D3 = 101) will reset the transmit buffer empty interrupt.

For character oriented protocols (Bisync, Monosync), the same discussion is valid, except that during transmit underrun condition and transmit underrun/EOM bit in set state, instead of flags, filler sync characters are transmitted.

CRC Generation

The transmit CRC enable bit (WR5: D0) must be set before loading any data into the MPSC. The CRC generator must be reset to all 1's at the beginning of each frame before CRC computation has begun. The CRC computation starts on the first data character loaded from the CPU and continues until the last data character. The CRC generated is inverted before it is sent on the Tx Data line.

Transmit Termination

A successful transmission can be terminated by issuing a "Reset Transmit Interrupt/DMA Pending" command, as discussed earlier. However, the transmitter may be disabled any time during the transmission and the results will be as shown in Figure 12.

RECEIVE IN INTERRUPT MODE

The receiver has to be initialized into the appropriate receive mode (see sample program later in this application note). The receiver must be programmed into Hunt Mode (WR3: D4) before it is enabled (WR3: D0). The receiver will remain in the Hunt Mode until a flag (or sync character) is received. While in the SDLC/Bisync/Monosync mode, the receiver does not enter the Hunt Mode unless the Hunt bit (WR3, D4) is set again or the receiver is enabled again.

SDLC Address byte is stored in WR6. A global address (FFH) has been hardwired on the MPSC. In address search mode (WR3: D2 = 1), any frame with address matching with the address in WR6 will be received by the MPSC. Frames with global address (FFH) will also be received, irrespective of the condition of address search mode bit (WR3: D2). In general receive mode (WR3: D2 = 0), all frames will be received.

Transmitter Disabled during	Result
1. Data Transmission	Tx Data will send idle characters* which will be zero inserted.
2. CRC Transmission	16 bit transmission, corresponding to 16 bits of CRC will be completed. However, flag bits will be substituted in the CRC field.
3. Immediately after issuing ABORT command.	Abort will still be transmitted—output will be in the mark state.

Figure 12. Transmitter Disabled During Transmission

*NOTE:
Idle characters are defined as a string of 15 or more contiguous ones.

Since the MPSC only recognizes single byte address field, extended address recognition will have to be done by the CPU on the data passed on by the MPSC. If the first address byte is checked by the MPSC, and the CPU determines that the second address byte does not have the correct address field, it must set the Hunt Mode (WR3: D2 = 1) and the MPSC will start searching for a new address byte preceded by a flag.

Programmable Interrupts

The receiver may be programmed into any one of the four modes. See Figure 13 for details.

WR1, CHA D4	WR1, CHA D3	Rx Interrupt Mode
0	0	Rx INT/DMA disable
0	1	Rx INT on first character
1	0	INT on all Rx characters (Parity affects vector)
1	1	INT on all Rx characters (Parity does not affect vector)

Figure 13. Receiver Interrupt Modes

All receiver interrupts can be disabled by WR1: D4 D3 = 00. Receiver interrupt on first character is normally used to start a DMA transfer or a block transfer sequence using WAIT to synchronize the data transfer to received or transmitted data.

External Status Interrupts

Any change in $\overline{CD}$ input or Abort detection in the received data, will generate an interrupt if External Status Interrupt was enabled (WR1: D0).

Special Receive Conditions

The receiver buffer is quadruply buffered. If the CPU fails to respond to "receive character" available interrupt within a period of three byte times (received bytes), the receiver buffer will overflow and generate an interrupt. Finally, at the end of the received frame, an interrupt will be generated when a valid ending flag has been detected.

Receive Character Length

The receive character length (6, 7 or 8 bits/character) may be changed during reception. However, to ensure that the change is effective on the next received character, this must be done fast enough such that the bits specified for the next character have not been assembled.

CRC Checking

The opening flag in the frame resets the receive CRC generator and any field between the opening and closing flag is checked for the CRC. In case of a CRC error, the CRC/Framing Error bit in status register 1 is set (RR1: D6 = 1). Receiver CRC may be disabled/enabled by WR3,D3. The CRC bytes on the received frame are passed on to the CPU just like data, and may be discarded by the CPU.

Receive Terminator

An end of frame is indicated by End of Frame interrupt. The CPU may issue an "Error Reset" command to reset this interrupt.

DMA (Direct Memory Access) Mode

The 8274 can be interfaced directly to the Intel DMA Controllers 8237A, 8257A and Intel I/O Processor 8089. The 8274 can be programmed into DMA mode by setting appropriate bits in WR2A. See Figure 11 for details.

TRANSMIT IN DMA MODE

After initializing the 8274 into the DMA mode, the first character must be loaded from the CPU to start the DMA cycle. When the first data character (may be the address byte in SDLC) is transferred from the transmit buffer to the transmit serial shift register, the transmit buffer gets empty and a transmit DMA request (TxDRQ) is generated for the channel. Just like the interrupt mode, to ensure that the CRC bytes are included in the frame, the transmit under-run/EOM latch must be reset. This should preferably be done after loading the first character from the CPU. The DMA will progress without any CPU intervention. When the DMA controller reaches the terminal count, it will not respond to the DMA request, thus letting the MPSC under-run. This will ensure CRC transmission. However, the under-run condition will generate an interrupt due to the Tx under-run/EOM bit getting set (RR0: D6). The CPU should issue a "Reset TxInt/DRQ pending" command to reset TxDRQ and issue a "Reset External Status" command to reset Tx Under-run/EOM interrupt. Following the CRC transmission, flag (7EH) will be loaded into the transmit buffer. This will also generate the TxDRQ since the transmit buffer is empty following the transmission of the CRC bytes. The CPU may issue a "Reset TxINT/DRQ pending" command to reset the TxDRQ. "Reset TxINT/DRQ pending" command must be issued before setting up the transmit DMA channel on the DMA Controller, otherwise the MPSC will start the DMA transfer immediately after the DMA channel is set up.

RECEIVE IN DMA MODE

The receiver must be programmed in RxINT on first receive character mode (WR1: D4 D3 = 0 1). Upon receiving the first character, which may be the address byte in SDLC, the MPSC generates an interrupt and also generates a Rx DMA Request (Rx DRQ) for the appropriate channel. The CPU has three byte times to service this interrupt (enable the DMA controller, etc.) before the receiver buffer will overflow. It is advisable to initialize the DMA controller before receiving the first character. In case of high bit rates, the CPU will have to service the interrupt very fast in order to avoid receiver over-run.

Once the DMA is enabled, the received data is transferred to the memory under DMA control. Any received error conditions or external status change condition will generate an interrupt as in the interrupt driven mode. The End of Frame is indicated by the End of Frame interrupt which is generated on reception of the closing flag of the SDLC frame. This End of Frame condition also disables the Receive DMA request. The End of Frame interrupt may be reset by issuing an "Error Reset" command to the MPSC. The "Error Reset" command also re-enables the Receive DMA request. It may be noted that the End of Frame condition sets bit D7 in RR1. This bit gets reset by "Error Reset" command. However, End of Frame bit (RR1: D7) can also be reset by the flag of the next incoming frame. For proper operation, Error Reset Command should be issued "after" the End of Frame Bit (RR1: D7) is set. In a more general case, "Error Reset" command should be issued after End of Frame, Receive over-run or Receive parity bit are set in RR1.

Wait Mode

The wait mode is normally used for block transfer by synchronizing the data transfer through the Ready output from the MPSC, which may be connected to the Ready input of the CPU. The mode can be programmed by WR 1, D7 D5 and may be programmed separately and independently on CH A and CH B. The Wait Mode will be operative if the following conditions are satisfied.

(i) Interrupts are enabled.

(ii) Wait Mode is enabled (WR1: D7)

(iii) CS = 0, A1 = 0

The RDY output becomes active when the transmitter buffer is full or receiver buffer is empty. This way the RDY output from the MPSC can be used to extend the CPU read and write cycle by inserting WAIT states. RDY_A or RDY_B are in high impedance state when the corresponding channel is not selected. This makes it possible to connect RDY_A and RDY_B outputs in wired OR configuration. Caution must be exercised here in using the RDY outputs of the MPSC or else the CPU may hang up for indefinite period. For example, let us assume that transmitter buffer is full and RDY_A is active, forcing the CPU into a wait state. If the $\overline{CTS}$ goes inactive during this period, the RDY_A will remain active for indefinite period and CPU will continue to insert wait states.

Vectored/Non-Vectored Mode

The MPSC is capable of providing an interrupt vector in response to the interrupt acknowledge sequence from the CPU. WR2, CH B contains this vector and the vector can be read in status register RR2. WR2, CH A (bit D5) can program the MPSC in vectored or non-vectored mode. See Figure 14 for details.

In both cases, WR2 may still have the vector stored in it. However, in vectored mode, the MPSC will put the vector on the data bus in response to the INTA (Interrupt Acknowledge) sequence as shown in Figure 15. In non-vectored mode, the MPSC will not respond to the INTA sequence. However, the CPU can read the vector by polling Status Register RR2. WR2A, D4 and D3 can be programmed to respond to 8085 or 8086 INTA sequence. It may be noted here that $\overline{IPI}$ (Interrupt Priority In) pin on the MPSC must be active for the vector to appear on the data bus.

WR2A, D5	Interrupt Mode
0	Non-vectored Interrupt
1	Vectored Interrupt

Figure 14. Vectored Interrupt

STATUS AFFECT VECTOR

The Vector stored in WR2B can be modified by the source of the interrupt. This can be done by setting the Status Affect Vector bit (WR1: D2). This powerful feature of the MPSC provides fast interrupt response time, by eliminating the need of writing a routine to read the status of the MPSC. Three bits of the vector are modified in eight different ways as shown on Figure 16. Bits V4, V3, V2 are modified in 8085 based system and bits V2, V1, V0 are modified in 8086/88 based system.

In non-vectored mode, the status affect vector mode can still be used and the vector read by the CPU. Status register RR2B (Read Register 2 in Channel B) will contain this modified vector.

WR2A D5	WR2A D4	WR2A D3	IPI	Mode	1st INTA	2nd INTA	3rd INTA
0	X	X	X	Non-Vectored	HI-Z	HI-Z	HI-Z
1	0	0	0	8085-1	1100 1101	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	0	1	8085-1	1100 1101	HI-Z	HI-Z
1	0	1	0	8085-2	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	1	1	8085-2	HI-Z	HI-Z	HI-Z
1	1	0	0	8086	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	—
1	1	0	1	8086	HI-Z	HI-Z	—

Figure 15. MPSC Vectored Interrupts

(8085) (8086)	V4 V2	V3 V1	V2 V0	Channel	Interrupt Source
	0	0	0	B	Tx Buffer Empty
	0	0	1		EXT/STAT Change
	0	1	0		RX CHAR Available
	0	1	1		Special Rx Condition
	1	0	0	A	Tx Buffer Empty
	1	0	1		EXT/STAT Change
	1	1	0		RX CHAR Available
	1	1	1		Special Rx Condition

Rx Special Condition: Parity Error, Framing Error, Rx Over-run Error, EOF (SDLC).
EXT/STAT Change: Change in Modem Control Pin Status: CTS, DCD, SYNC, EOM, Break/Abort Detection.

Figure 16. Status Affect Vector Mode

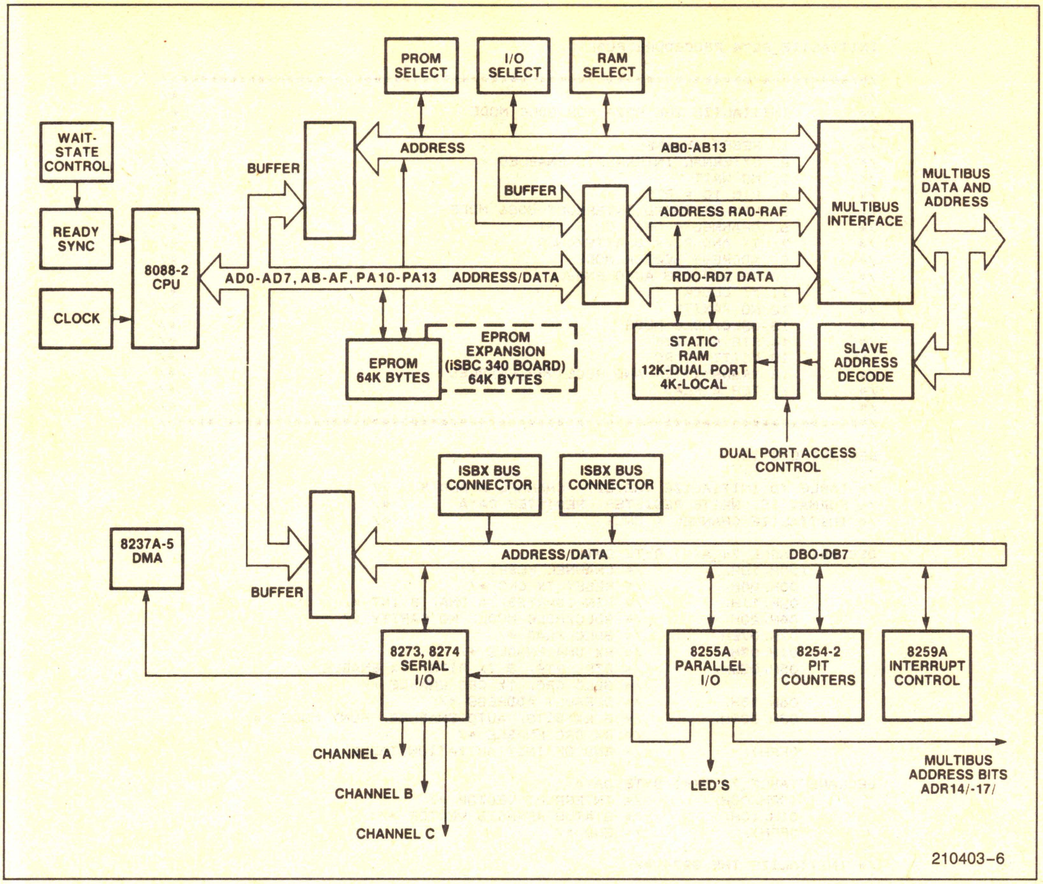


Figure 17. Functional Block Diagram—iSBC® 88/45

APPLICATION EXAMPLE

This section describes the hardware and software of an 8274/8088 system. The hardware vehicle used is the INTEL Single Board Computer iSBC 88/45—Advanced Communication Controller. The software which exercises the 8274 is written in PLM 86. This example will demonstrate how 8274 can be configured into the SDLC mode and transfer data through DMA control. The hardware example will help the reader configure his hardware and the software examples will help in developing an application software. Most software examples closely approximate real data link controller software in the SDLC communication and may be used with very little modification.

iSBC® 88/45

A brief description of the iSBC 88/45 board will be presented here. For more detailed information on the board and the schematics, refer to Hardware Manual for the iSBC 88/45, Advanced Communication Controller. iSBC 88/45 is an intelligent slave/multimaster communication board based on the 8088 processor, the 8274 and the 8273 SDLC/HDLC controller. Figure 17 shows the functional block diagram of the board. The iSBC 88/45 has the following features.

- 8 MHz processor
- 16K bytes of static RAM (12K dual port)
- Multimaster/Intelligent Slave Multibus Interface
- Nine Interrupt Levels 8259A
- Two serial channels through 8274
- One Serial channel through 8273
- S/W programmable baud rate generator
- Interfaces: RS232, RS422/449, CCITT V.24
- 8237A DMA controller
- Baud Rate to 800K Baud

```
INITIALIZE_8274: PROCEDURE PUBLIC;

/*************************************************************************/
/*                                                                       */
/*      INITIALIZE THE 8274 FOR SDLC MODE                                */
/*                                                                       */
/*      1.  RESET CHANNEL                                                */
/*      2.  EXTERNAL INTERRUPTS ENABLED                                  */
/*      3.  NO WAIT                                                      */
/*      4.  PIN 10 = RTS                                                 */
/*      5.  NON-VECTORED INTERRUPT-8086 MODE                             */
/*      6.  CHANNEL A DMA, CH B INT                                      */
/*      7.  TX AND RX = 8 BITS/CHAR                                      */
/*      9.  ADDRESS SEARCH MODE                                          */
/*      10. CD AND CTS AUTO ENABLE                                       */
/*      11. X1 CLOCK                                                     */
/*      12. NO PARITY                                                    */
/*      13. SDLC/HDLC MODE                                               */
/*      14. RTS AND DTR                                                  */
/*      15. CCITT - CRC                                                  */
/*      16. TRANSMITTER AND RECEIVER ENABLED                             */
/*      17. 7EH = FLAG                                                   */
/*                                                                       */
/*************************************************************************/

DECLARE C BYTE;

/* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B    */
/* FORMAT IS: WRITE REGISTER, REGISTER DATA        */
/* INITIALIZE CHANNEL A ONLY                       */

DECLARE TABLE_74_A(*) BYTE DATA
        (00H, 18H,          /* CHANNEL RESET */
         00H, 80H,          /* RESET TX CRC */
         02H, 11H,          /* PIN 10=RTSB, A DMA, B INT */
         04H, 20H,          /* SDLC/HDLC MODE, NO PARITY */
         07H, 07EH,         /* SDLC FLAG */
         01H, 08H,          /* RX DMA ENABLE */
         05H, 0EBH,         /* DTR, RTS, 8 TX BITS, TX ENABLE, */
                            /* SDLC CRC, TX CRC ENABLE */
         06H, 55H,          /* DEFAULT ADDRESS */
         03H, 0D9H,         /* 8 RX BITS, AUTO ENABLES, HUNT MODE, */
                            /* RX CRC ENABLE */
         0FFH);             /* END OF INITIALIZATION TABLE */

DECLARE TABLE_74_B(*) BYTE DATA
        (02H, 00H,          /* INTERRUPT VECTOR */
         01H, 1CH,          /* STATUS AFFECTS VECTOR */
         0FFH);             /* END */

/* INITIALIZE THE 8274 */

C=0;
DO WHILE TABLE_74_B(C) <> 0FFH;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
END;

C=0;
DO WHILE TABLE_74_A(C) <> 0FFH;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
END;
RETURN;
END INITIALIZE_8274;
```

210403-7

Figure 18. Typical MPSC SDLC Initialization Sequence

For this application, the CPU is run at 8 MHz. The board is configured to operate the 8274 in SDLC operation with the data transfer in DMA mode using the 8237A. 8274 is configured first in non-vectored mode in which case the INTEL Priority Interrupt Controller 8259A is used to resolve priority between various interrupting sources on the board and subsequently interrupt the CPU. However, the vectored mode of the 8274 is also verified by disabling the 8259A and reading the vectors from the 8274. Software examples for each case will be shown later.

The application example is interrupt driven and uses DMA for all data transfers under 8237A control. The 8254 provides the transmit and receive clocks for the 8274. The 8274 was run at 400K baud with a local loopback (jumper wire) on Channel A data. The board was also run at 800K baud by modifying the software as will be discussed later in the Special Applications section. One detail to note is that the Rx Channel DMA request line from the 8274 has higher priority than the Tx Channel DMA request line. The 8274 master clock was 4.0 MHz. The on-board RAM is used to define transmit and receive data buffers. In this application, the data is read from memory location 800H through 810H and transferred to memory location 900H to 910H through the 8274 Serial Link. The operation is full duplex. 8274 modem control pins, $\overline{CTS}$ and $\overline{CD}$ have been tied low (active).

Software

The software consists of a monitor program and a program to exercise the 8274 in the SDLC mode. Appendix A contains the entire program listing. For the sake of clarity, each source module has been rewritten in a simple language and will be discussed here individually. Note that some labels in the actual listings in the Appendix will not match with the labels here. Also the listing in the Appendix sets up some flags to communicate with the monitor. Some of these flags are not explained in detail for the reason that they are not pertinent to this discussion. The monitor takes the command from a keyboard and executes this program, logging any error condition which might occur.

8274 Initialization

The MPSC is initialized in the SDLC mode for Channel A. Channel B is disabled. See Figure 18 for the initialization routine. Note that WR4 is initialized before setting up the transmitter and receive parameters. However, it may also be pointed out that other than WR4, all the other registers may be programmed in any order. Also SDLC-CRC has been programmed for correct operation. An incorrect CRC selection will result in incorrect operation. Also note that receive interrupt on first receive character has been programmed although Channel A is in the DMA mode.

Interrupt Routines

The 8274 interrupt routines will be discussed here. On an 8274 interrupt, program branches off to the "Main Interrupt Routine". In main interrupt routine, status register RR2 is read. RR2 contains the modified vector. The cause of the interrupt is determined by reading the modified bits of the vector. Note that the 8274 has been programmed in the non-vectored mode and status affects vector bit has been set. Depending on the value of the modified bits, the appropriate interrupt routine is called. See Figure 19 for the flow diagram and Figure 20 for the source code. Note that an End of Interrupt Command is issued after servicing the interrupt. This is necessary to enable the lower priority interrupts.

Figure 21 shows all the interrupt routines called by the Main Interrupt Routine. "Ignore-Interrupt" as the name implies, ignores any interrupts and sets the FAIL flag. This is done because this program is for Channel A only and we are ignoring any Channel B interrupts. The important thing to note is the Channel A Receiver Character available routine. This routine is called after receiving the first character in the SDLC frame. Since the transfer mode is DMA, we have a maximum of three character times to service this interrupt by enabling the DMA controller.

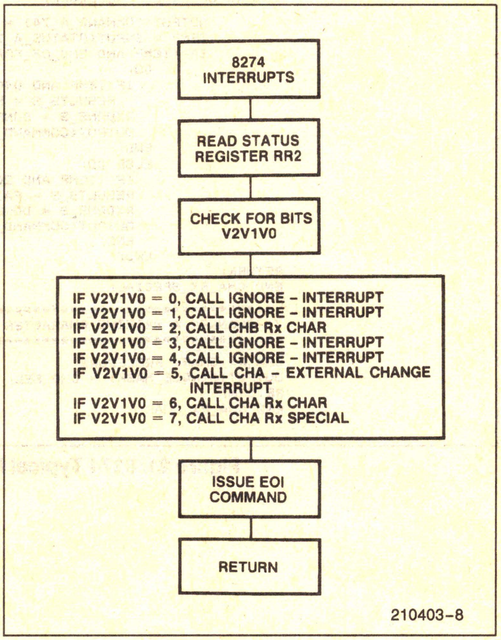


Figure 19. Interrupt Response Flow Diagram

```
/****************************/
/* MAIN INTERRUPT ROUTINE */
/****************************/

OUTPUT(COMMAND_B_74) = 2;                  /* SET POINTER TO 2*/
TEMP = INPUT(STATUS_B_74) AND 07H;         /* READ INTERRUPT VECTOR */
                                           /* CHECK FOR CHA INT ONLY*/
/* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
DO CASE TEMP;
        CALL    IGNORE_INT;                /* V2V1V0 = 000*/
        CALL    IGNORE_INT;                /* V2V1V0 = 001*/
        CALL    CHB_RX_CHAR;               /* V2V1V0 = 010*/
        CALL    IGNORE_INT;                /* V2V1V0 = 011*/
        CALL    IGNORE_INT;                /* V2V1V0 = 100*/
        CALL    CHA_EXTERNAL_CHANGE;       /* V2V1V0 = 101*/
        CALL    CHA_RX_CHAR;               /* V2V1V0 = 110*/
        CALL    CHA_RX_SPECIAL;            /* V2V1V0 = 111*/
END;
OUTPUT(COMMAND_A_74) =38H;        /* END OF INTERRUPT FOR 8274 */
RETURN;
END INTERRUPT_8274;
```

210403-9

Figure 20. Typical Main Interrupt Routine

```
/**********************************************************/
/* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
/**********************************************************/

CHA_EXTERNAL_CHANGE: PROCEDURE;

TEMP = INPUT(STATUS_A_74);        /* STATUS REG 1*/
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
        TXDONE_S=DONE;
ELSE DO;
        TXDONE_S=DONE;
        RESULTS_S=FAIL;
    END;
OUTPUT(COMMAND_A_74) = 10H;       /* RESET EXT/STATUS INTERRUPTS */
RETURN;
END CHA_EXTERNAL_CHANGE;
/**********************************************************/
/* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
/**********************************************************/

CHA_RX_SPECIAL: PROCEDURE;

        OUTPUT(COMMAND_A_74) = 1;
        TEMP = INPUT(STATUS_A_74);
        IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
              DO;
                IF(TEMP AND 040H) = 040H THEN
                  RESULTS_S = FAIL;         /* CRC ERROR */
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
              END;
              ELSE DO;
                IF (TEMP AND 20H) = 20H THEN DO;
                RESULTS_S = FAIL;           /* RX OVERRUN ERROR*/
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
                END;
              END;
RETURN;
END CHA_RX_SPECIAL;
/******************************************/
/* CHANNEL A RECEIVE CHARACTER AVAILABLE */
/******************************************/

CHA_RX_CHAR: PROCEDURE;
OUTPUT(SINGLE_MASK) = CHO_SEL;              /*ENABLE RX DMA CHANNEL*/
RETURN;
END CHA_RX_CHAR;
```

210403-10

Figure 21. 8274 Typical Interrupt Handling Routines

It may be recalled that the receiver buffer is three bytes deep in addition to the receiver shift register. At very high data rates, it may not be possible to have enough time to read RR2, enable the DMA controller without overrunning the receiver. In a case like this, the DMA controller may be left enabled before receiving the Receive Character Interrupt. Remember, the Rx DMA request and interrupt for the receive character appears at the same time. If the DMA controller is enabled, it would service the DMA request by reading the received character. This will make the 8274 interrupt line go inactive. However, the 8259A has latched the interrupt and a regular interrupt acknowledge sequence still occurs after the DMA controller has completed the transfer and given up the bus. The 8259A will return Level 7 interrupt since the 8274 interrupt has gone away. The user software must take this into account, otherwise the CPU will hang up.

The procedure shown for the Special Receive Condition Interrupt checks if the interrupt is due to the End of Frame. If this is not TRUE, the FAIL flag is set and the program aborted. For a real life system, this must be followed up by error recovery procedures which obviously are beyond the scope of this Application Note.

The transmission is terminated when the End of Message (RR0, D6) interrupt is generated. This interrupt is serviced in the Channel A External/Status Change interrupt procedure. For any other change in external status conditions, the program is aborted and a FAIL flag set.

Main Program

Finally, we will briefly discuss the main program. Figure 22 shows the source program. It may be noted that the Transmit Under-run latch is reset after loading the first character into the 8274. This is done to ensure CRC transmission at the end of the frame. Also, the first character is loaded from the CPU to start DMA transfer of subsequent data. This concludes our discussion on hardware and software example. Appendix A also includes the software written to exercise the 8274 in the vectored mode by disabling the 8259A.

```
CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

        CALL    ENABLE_INTERRUPTS_S;
        CALL    INIT_8274_SDLC_S;
        ENABLE;
        OUTPUT(COMMAND_A_74) = 28H;   /* RESET TX INT/DMA          */
        OUTPUT(COMMAND_B_74) = 28H;   /* BEFORE INITIALIZING 8237*/
        CALL    INIT_8237_S;
        OUTPUT(DATA_A_74) = 55H;      /*LOAD FIRST CHARACTER FROM */
                                      /*CPU */
        /* TO ENSURE CRC TRANSMISSION, RESET TX UNDERRUN LATCH    */
        OUTPUT(COMMAND_A_74) = 0C0H;
        RXDONE_S,TXDONE_S=NOT_DONE;   /* CLEAR ALL FLAGS           */
        RESULTS_S=PASS;               /* FLAG SET FOR MONITOR      */
        DO WHILE TXDONE_S=NOT_DONE;   /* DO UNTIL TERMINAL COUNT  */
        END;

        DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
        /* WAIT FOR CRC TO GET TRANSMITTED */
        /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
        END;
        DO WHILE RXDONE_S=NOT_DONE;   /* DO UNTIL TERMINAL COUNT  */
        END;
        CALL    STOP_8237_S;
END CHA_SDLC_TEST;
```

210403-11

Figure 22. Typical 8274 Transmit/Receive Set-Up in SDLC Mode

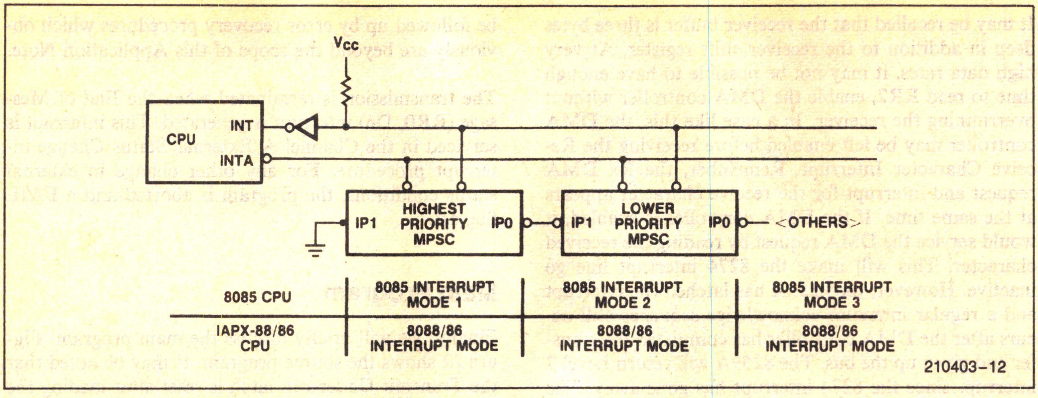


Figure 23. 8274 Daisy Chain Vectored Mode

SPECIAL APPLICATIONS

In this section, some special application issues will be discussed. This will be useful to a user who may be using a mode which is possible with the 8274 but not explicitly explained in the data sheet.

MPSC Daisy Chain Operation

Multiple MPSCs can be connected in a daisy-chain configuration (see Figure 23). This feature may be useful in an application where multiple communication channels may be required and because of high data rates, conventional interrupt controller is not used to avoid long interrupt response times. To configure the MPSCs for the daisy chain operation, the interrupt priority input pins ($\overline{\text{IPI}}$) and interrupt priority output pins ($\overline{\text{IPO}}$) of the MPSC should be connected as shown. The highest priority device has its $\overline{\text{IPI}}$ pin connected to ground. Each MPSC is programmed in a vectored mode with status affects vector bit set. In the 8085 basic systems, only one MPSC should be programmed in the 8085 Mode 1. This is the MPSC which will put the call vector (CD Hex) on the data bus in response to the first $\overline{\text{INTA}}$ pulse (see Figure 15). It may be pointed out that the MPSC in 8085 Mode 1 will provide the call vector irrespective of the state of $\overline{\text{IPI}}$ pin. Once a higher priority MPSC generates an interrupt, its $\overline{\text{IPO}}$ pin goes inactive thus preventing lower priority MPSCs from interrupting the CPU. Preferably the highest priority MPSC should be programmed in 8085 Mode 1. It may be recalled that the Priority Resolve Time on a given MPSC extends from the falling edge of the first $\overline{\text{INTA}}$ pulse to the falling edge of the second $\overline{\text{INTA}}$ pulse. During this period, no new internal interrupt requests are accepted. The maximum number of the MPSCs that can be connected in a daisy chain is limited by the Priority Resolution Time. Figure 24 shows a maximum number of MPSCs that can be connected in various CPU systems.

It may be pointed out that $\overline{\text{IOP}}$ to $\overline{\text{IPI}}$ delay time specification is 100 ns.

System Configuration	Priority Resolution Time Min (ns)	Number of 8274s Daisy Chained (Max)
8086-1	400	4
8086-2	500	5
8086	800	8
8088	800	8
8085-2	1200	12
8085A	1920	19

NOTE:
Zero wait states have been assumed.

Figure 24. 8274 Daisy Chain Operation

Bisync Transparent Communication

Bisync applications generally require that data transparency be established during communication. This requires that the special control characters may not be included in the CRC accumulation. Refer to the Synchronous Protocol Overview section for a more detailed discussion on data transparency. The 8274 can be used for transparent communication in Bisync communications. This is made possible by the capability of the MPSC to selectively turnon/turnoff the CRC accumulation while transmitting or receiving. In bisync transparent transmit mode, the special characters (DLE, DLE SYN, etc) are excluded from CRC calculation. This can be easily accomplished by turning off the transmit CRC calculation (WR5: D5 = 0) before loading the special character into the transmit buffer. If the next character is to be included in the CRC accumulation, then the CRC can be enabled (WR5: D5 = 1). See Figure 25 for a typical flow diagram.

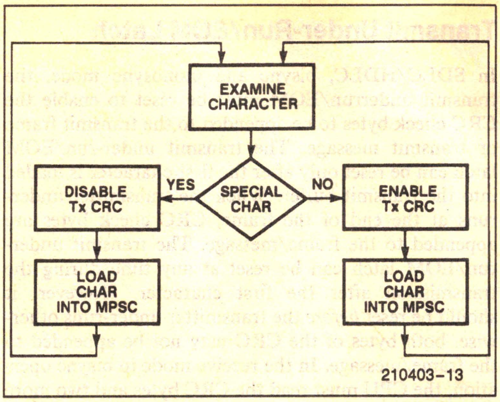


Figure 25. Transmit In Bisync Transparent Mode

During reception, it is possible to exclude received character from CRC calculation by turning off the Receive CRC after reading the special character. This is made possible by the fact that the received data is presented to receive CRC checker 8 bit times after the character has been received. During this 8 bit times, the CPU must read the character and decide if it wants to be included in the CRC calculation. Figure 26 shows the typical flow diagram to achieve this.

It should be noted that the CRC generator must be enabled during CRC reception. Also, after reading the CRC bytes, two more characters (SYNC) must be read before checking for CRC check result in RR1.

Auto Enable Mode

In some data communication applications, it may be required to enable the transmitter or the receiver when the $\overline{CTS}$ or the $\overline{CD}$ lines respectively, are activated by the modems. This may be done very easily by programming the 8274 into the Auto Enable Mode. The auto enable mode is set by writing a '1' to WR3,D5. The function of this mode is to enable the transmitter automatically when $\overline{CTS}$ goes active. The receiver is enabled when $\overline{CD}$ goes active. An in-active state of $\overline{CTS}$ or $\overline{CD}$ pin will disable the transmitter or the receiver respectively. However, the Transmit Enable bit (WR5:D3) and Receive Enable bit (WR3:D1) must be set in order to use the auto enable mode. In non-auto mode, the transmitter or receiver is enabled if the corresponding bits are set in WR5 and WR3, irrespective of the state $\overline{CTS}$ or $\overline{CD}$ pins. It may be recalled that any transition on $\overline{CTS}$ or $\overline{CD}$ pin will generate External/Status Interrupt with the corresponding bits set in RR1. This interrupt can be cleared by issuing a Reset External/Status interrupt command as discussed earlier.

Note that in auto enable mode, the character to be transmitted must be loaded into the transmit buffer after the $\overline{CTS}$ becomes active, not before. Any character loaded into the transmit buffer before the $\overline{CTS}$ became active will not be transmitted.

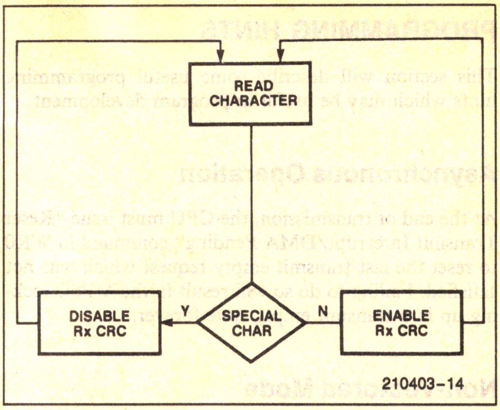


Figure 26. Receive In Bisync Transparent Mode

High Speed DMA Operation

In the section titled Application Example, the MPSC has been programmed to operate in DMA mode and receiver is programmed to generate an interrupt on the first receive character. You may recall that the receive FIFO is three bytes deep. On receiving the interrupt on the first receive character, the CPU must enable the DMA controller within three received byte times to avoid receiver over-run condition. In the application example, at 400K baud, the CPU had approximately 60 μs to enable the DMA controller to avoid receiver buffer overflow. However, at higher baud rates, the CPU may not have enough time to enable the DMA controller in time. For example, at 1M baud, the CPU should enable the DMA controller within approximately 24 μs to avoid receiver buffer overrun. In most applications, this is not sufficient time. To solve this problem, the DMA controller should be left enabled before getting the interrupt on the first receive character (which is accompanied by the Rx DMA request for the appropriate channel). This will allow he DMA controller to start DMA transfer as soon as the Rx DMA request becomes active without giving the CPU enough time to respond to the interrupt on the first receive character. The CPU will respond to the interrupt *after* the DMA transfer has been completed and will find the 8259A (see Application Example) responding with interrupt level 7, the lowest priority level. Note that the 8274 interrupt request was satisfied by the DMA controller, hence the interrupt on the first receive character was cleared and the 8259A had no pending interrupt. Because of no pending interrupt, the 8259A returned interrupt level 7 in response to the INTA sequence from the CPU. The user software should take care of this interrupt.

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

Status Register RR2

ChB RR2 contains the vector which gets modified to indicate the source of interrupt (see the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of ChB RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-Run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit underrun/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four singals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

APPENDIX A
APPLICATION EXAMPLE: SOFTWARE LISTINGS

```
PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8274_S
OBJECT MODULE PLACED IN :F1:SINI74.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI74.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

              /***************************************************/
              /*                                                 */
              /*       INITIALIZE THE 8274 FOR SDLC MODE         */
              /*                                                 */
              /*       1. RESET CHANNEL                          */
              /*       2. EXTERNAL INTERRUPTS ENABLED            */
              /*       3. NO WAIT                                */
              /*       4. PIN 10 = RTS                           */
              /*       5. NON-VECTORED INTERRUPT-8086 MODE       */
              /*       6. CHANNEL A DMA, CH B INT                */
              /*       7. TX AND RX = 8 BITS/CHAR                */
              /*       9. ADDRESS SEARCH MODE                    */
              /*       10. CD AND CTS AUTO ENABLE                */
              /*       11. X1 CLOCK                              */
              /*       12. NO PARITY                             */
              /*       13. SDLC/HDLC MODE                        */
              /*       14. RTS AND DTR                           */
              /*       15. CCITT - CRC                           */
              /*       16. TRANSMITTER AND RECEIVER ENABLED      */
              /*       17. 7EH = FLAG                            */
              /*                                                 */
              /***************************************************/

  1           INIT_8274_S: DO;

              $INCLUDE (:F1:PORTS.PLM)

          =   /**********************************************/
          =   /*                                            */
          =   /*      iSBC 88/45 PORT ASSIGNMENTS           */
          =   /*                                            */
          =   /**********************************************/

  2   1   =   DECLARE LIT LITERALLY 'LITERALLY';

          =   /* 8237A-5  PORTS */

  3   1   =   DECLARE CHO_ADDR        LIT     '080H',
          =           CHO_COUNT       LIT     '081H',
          =           CH1_ADDR        LIT     '082H',
          =           CH1_COUNT       LIT     '083H',
          =           CH2_ADDR        LIT     '084H',
          =           CH2_COUNT       LIT     '085H',
          =           CH3_ADDR        LIT     '086H',
          =           CH3_COUNT       LIT     '087H',
          =           STATUS_37       LIT     '088H',
          =           COMMAND_37      LIT     '088H',
          =           REQUEST_REG_37  LIT     '089H',
          =           SINGLE_MASK     LIT     '08AH',
          =           MODE_REG_37     LIT     '08BH',
```



```
          =           CLR_BYTE_PTR_37 LIT     '08CH',
          =           TEMP_REG_37     LIT     '08DH',
          =           MASTER_CLEAR_37 LIT     '08DH',
          =           ALL_MASK_37     LIT     '08FH';

          =   /* 8254-2 PORTS */

  4   1   =   DECLARE CTR_00          LIT     '090H',
          =           CTR_01          LIT     '091H',
          =           CTR_02          LIT     '092H',
```

210403–15

```
                =           CONTROLO_54     LIT      '093H',
                =           STATUSO_54      LIT      '093H',
                =           CTR_10          LIT      '098H',
                =           CTR_11          LIT      '099H',
                =           CTR12           LIT      '09AH',
                =           CONTROL1_54     LIT      '09BH',
                =           STATUS1_54      LIT      '09BH';

                =     /* 8255 PORTS */

    5    1      =     DECLARE PORTA_55      LIT      '0A0H',
                =             PORTB_55      LIT      '0A1H',
                =             PORTC_55      LIT      '0A2H',
                =             CONTROL_55    LIT      '0A3H';

                =     /* 8274 PORTS */

    6    1      =     DECLARE DATA_A_74     LIT      '0D0H',
                =             DATA_B_74     LIT      '0D1H',
                =             STATUS_A_74   LIT      '0D2H',
                =             COMMAND_A_74  LIT      '0D2H',
                =             STATUS_B_74   LIT      '0D3H',
                =             COMMAND_B_74  LIT      '0D3H';

                =     /* 8259A PORTS */

    7    1      =     DECLARE STATUS_POLL_59 LIT     '0E0H',
                =             ICW1_59       LIT      '0E0H',
                =             OCW2_59       LIT      '0E0H',
                =             OCW3_59       LIT      '0E0H',
                =             OCW1_59       LIT      '0E1H',
                =             ICW2_59       LIT      '0E1H',
                =             ICW3_59       LIT      '0E1H',
                =             ICW4_59       LIT      '0E1H';

                =     /* 8274 REGISTER BIT ASSIGNMENTS */
                =     /* READ REGISTER 0 */

    8    1      =     DECLARE RX_AVAIL      LIT      '01H',
                =             INT_PENDING   LIT      '02H',
                =             TX_EMPTY      LIT      '04H',
                =             CARRIER_DETECT LIT     '08H',
                =             SYNC_HUNT     LIT      '10H',
                =             CLEAR_TO_SEND LIT      '20H',

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

                =             END_OF_TX_MESSAGE LIT  '40H',
                =             BREAK_ABORT   LIT      '80H';

                =     /* READ REGISTER 1 */

    9    1      =     DECLARE ALL_SENT      LIT      '01H',
                =             PARITY_ERROR  LIT      '10H',
                =             RX_OVERRUN    LIT      '20H',
                =             CRC_ERROR     LIT      '40H',
                =             END_OF_FRAME  LIT      '80H';

                =     /* READ REGISTER 2 */

   10    1      =     DECLARE TX_B_EMPTY    LIT      '00H',
                =             EXT_B_CHANGE  LIT      '01H',
                =             RX_B_AVAIL    LIT      '02H',
                =             RX_B_SPECIAL  LIT      '03H',
                =             TX_A_EMPTY    LIT      '04H',
                =             EXT_A_CHANGE  LIT      '05H',
                =             RX_A_AVAIL    LIT      '06H',
                =             RX_A_SPECIAL  LIT      '07H';
```

210403-16

```
          =     /* 8237 BIT ASSIGNMENTS */

 11   1   =     DECLARE CH0_SEL           LIT       '00H',
          =             CH1_SEL           LIT       '01H',
          =             CH2_SEL           LIT       '02H',
          =             CH3_SEL           LIT       '03H',
          =             WRITE_XFER        LIT       '04H',
          =             READ_XFER         LIT       '08H',
          =             DEMAND_MODE       LIT       '00H',
          =             SINGLE_MODE       LIT       '40H',
          =             BLOCK_MODE        LIT       '80H',
          =             SET_MASK          LIT       '04H';

 12   1         DELAY_S: PROCEDURE PUBLIC;
 13   2         DECLARE D WORD;
 14   2         D=0;
 15   2         DO WHILE D<800H;
 16   3         D=D+1;
 17   3         END;
 18   2         END DELAY_S;

 19   1         INIT_8274_SDLC_S:    PROCEDURE PUBLIC;

 20   2         DECLARE C    BYTE;

                $EJECT
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

                /* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B */

                /*  FORMAT IS: WRITE REGISTER, REGISTER DATA   */
                /*      INITIALIZE CHANNEL ONLY                 */

 21   2         DECLARE TABLE_74_A(*) BYTE DATA
                        (00H, 18H,          /* CHANNEL RESET */
                        00H, 80H,           /* RESET TX CRC */
                        02H, 11H,           /* PIN 10=RTSB, A DMA, B INT */
                        04H, 20H,           /* SDLC/HDLC MODE, NO PARITY */
                        07H, 07EH,          /* SDLC FLAG */
                        01H, 0BH,           /* RX DMA ENABLE */
                        05H, 0EBH,          /* DTR, RTS, 8 TX BITS, TX ENABLE, TX CRC ENABLE */
                        06H, 55H,           /* DEFAULT ADDRESS */
                        03H, 0D9H,          /* 8 RX BITS, AUTO ENABLES, HUNT MODE,  */
                                            /* RX CRC ENABLE */
                        0FFH);              /* END OF INITIALIZATION TABLE */

 22   2         DECLARE TABLE_74_B(*) BYTE DATA
                        (02H, 00H,          /* INTERRUPT VECTOR */
                        01H, 1CH,           /* STATUS AFFECTS VECTOR */
                        0FFH);              /* END */


                /* INITIALIZE THE 8254 */

 23   2         OUTPUT(CONTROL0_54)=36H;
 24   2         OUTPUT(CTR_00) = LOW(20);          /* BAUD RATE = 400K_BAUD*/
 25   2         OUTPUT(CTR_00) = HIGH(20);         /* BAUD RATE = 400K_BAUD*/

                /* INITIALIZE THE 8274 */

 26   2         C=0;
 27   2         DO WHILE TABLE_74_B(C) <> 0FFH;
 28   3             OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
 29   3             C=C+1;
 30   3             OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
 31   3             C=C+1;
 32   3         END;
```

210403-17

```
 33   2        C=0;
 34   2        DO WHILE TABLE_74_A(C) <> 0FFH;
 35   3            OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
 36   3            C=C+1;
 37   3            OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
 38   3            C=C+1;
 39   3        END;
 40   2            CALL    DELAY_S;

 41   2        RETURN;
 42   2        END INIT_8274_SDLC_S;
 43   1        END INIT_8274_S;




PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


MODULE INFORMATION:

     CODE AREA SIZE     = 00A8H     168D
     CONSTANT AREA SIZE = 0000H       0D
     VARIABLE AREA SIZE = 0003H       3D
     MAXIMUM STACK SIZE = 0006H       6D
     213 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION



PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8237_CHA
OBJECT MODULE PLACED IN :F1:SINI37.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI37.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC
TEST) COMPACT NOINTVECTOR ROM

               /*************************************************************************/
               /*                                                                       */
               /*      8237    INITIALIZATION ROUTINE   FOR DMA TRANSFER                */
               /*                                                                       */
               /*************************************************************************/

  1            INIT_8237_CHA: DO;

               $NOLIST

 12   1        INIT_8237_S: PROCEDURE PUBLIC;


 13   2        OUTPUT(MASTER_CLEAR_37)=0;
 14   2        OUTPUT(COMMAND_37) = 20H;        /* EXTENDED WRITE */
 15   2        OUTPUT(ALL_MASK_37) = 0FH;       /* MASK ALL REQUESTS */
 16   2        OUTPUT(MODE_REG_37) = (SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
 17   2        OUTPUT(MODE_REG_37) = (SINGLE_MODE OR READ_XFER OR CH1_SEL);
 18   2        OUTPUT(CLR_BYTE_PTR_37) = 0;
 19   2        OUTPUT(CH0_ADDR) = 00;         /* RECEIVE BUFF AT 900H */
 20   2        OUTPUT(CH0_ADDR) = 09H;
 21   2        OUTPUT(CH0_COUNT) = 0H;
 22   2        OUTPUT(CH0_COUNT) = 01;
 23   2        OUTPUT(CH1_ADDR) = 00;         /* TRANSMIT BUFF AT 800H */
 24   2        OUTPUT(CH1_ADDR) = 08H;
 25   2        OUTPUT(CH1_COUNT) = 010H;
 26   2        OUTPUT(CH1_COUNT) = 00H;
```

210403-18

```
                /* ENABLE TRANSFER */
 27   2         OUTPUT(SINGLE_MASK) = CH1_SEL;     /* ENABLE TX DMA */
 28   2         RETURN;

 29   2         END INIT_8237_S;

                /* TURN OFF THE 8237 CHANNELS 0 AND 1 */

 30   1         STOP_8237_S: PROCEDURE PUBLIC;
 31   2         OUTPUT(SINGLE_MASK) = CH1_SEL OR SET_MASK;
 32   2         OUTPUT(SINGLE_MASK) = CHO_SEL OR SET_MASK;
 33   2         RETURN;
 34   2         END STOP_8237_S;
 35   1         END INIT_8237_CHA;


MODULE INFORMATION:

     CODE AREA SIZE     = 004CH     76D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0000H      0D



PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


     MAXIMUM STACK SIZE = 0002H      2D
     163 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION


PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INTR_8274_S
OBJECT MODULE PLACED IN :F1:SINTR.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINTR.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

                /*********************************************/
                /*                                           */
                /*       8274 INTERRUPT ROUTINE              */
                /*                                           */
                /*********************************************/

  1             INTR_8274_S: DO;
                $NOLIST
 12   1         DECLARE TEMP BYTE;
 13   1         DECLARE (RESULTS_S,TXDONE_S,RXDONE_S) BYTE EXTERNAL;
 14   1         DECLARE INT_VEC POINTER AT (140);
 15   1         DECLARE INT_VEC_STORE POINTER;
 16   1         DECLARE MASK_59 BYTE;
 17   1         DECLARE DONE         LIT     'OFFH',
                        NOT_DONE     LIT     'OOH',
                        PASS         LIT     'OFFH',
                        FAIL         LIT     'OOH';

                /*****************************/
                /* IGNORE INTERRUPT HANDLER */
                /*****************************/

 18   1         IGNORE_INT: PROCEDURE;

 19   2         RESULTS_S = FAIL;
 20   2         RETURN;
 21   2         END IGNORE_INT;
```

210403-19

```
                 /*******************************************************/
                 /* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
                 /*******************************************************/

  22   1         CHA_EXTERNAL_CHANGE: PROCEDURE;

  23   2         TEMP = INPUT(STATUS_A_74);          /* STATUS REG 1*/
  24   2         IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
  25   2              TXDONE_S=DONE;
  26   2         ELSE DO;
  27   3              TXDONE_S=DONE;
  28   3              RESULTS_S=FAIL;
  29   3               END;
  30   2         OUTPUT(COMMAND_A_74) = 10H;  /* RESET EXT/STATUS INTERRUPTS */
  31   2         RETURN;
  32   2         END CHA_EXTERNAL_CHANGE;

                 $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

                 /**********************************************************/
                 /* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
                 /**********************************************************/

  33   1         CHA_RX_SPECIAL: PROCEDURE;

  34   2              OUTPUT(COMMAND_A_74) = 1;
  35   2              TEMP = INPUT(STATUS_A_74);
  36   2              IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
  37   2                        DO;
  38   3                          IF(TEMP AND 040H) = 040H THEN
  39   3                            RESULTS_S = FAIL;           /* CRC ERROR */
  40   3                          RXDONE_S = DONE;
  41   3                          OUTPUT(COMMAND_A_74) = 30H;  /*ERROR RESET*/
  42   3                        END;
  43   2                        ELSE DO;
  44   3                          IF (TEMP AND 20H) = 20H THEN DO;
  46   4                          RESULTS_S = FAIL;            /* RX OVERRUN ERROR*/
  47   4                          RXDONE_S = DONE;
  48   4                          OUTPUT(COMMAND_A_74) = 30H;  /*ERROR RESET*/
  49   4                          END;
  50   3                        END;
  51   2         RETURN;
  52   2         END CHA_RX_SPECIAL;


                 /******************************************/
                 /* CHANNEL A RECEIVE CHARACTER AVAILABLE */
                 /******************************************/

  53   1         CHA_RX_CHAR: PROCEDURE;
  54   2         OUTPUT(SINGLE_MASK) = CH0_SEL;            /*ENABLE RX DMA CHANNEL*/
  55   2         RETURN;
  56   2         END CHA_RX_CHAR;

                 $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

                 /* ENABLE 8274 INTERRUPTS - SET UP THE 8259A */

  57   1         ENABLE_INTERRUPTS_S: PROCEDURE PUBLIC;

  58   2         DECLARE CHA_INT_ON LIT '0F7H';

  59   2         DISABLE;

  60   2         CALL    SET$INTERRUPT(39, INT_39);
```

210403–20

```
61   2       INT_VEC_STORE = INT_VEC;
62   2       INT_VEC = INTERRUPT$PTR(INT_8274_S);
63   2       MASK_59 = INPUT(OCW1_59);

64   2       OUTPUT(OCW1_59) = MASK_59 AND CHA_INT_ON;

65   2       RETURN;
66   2       END ENABLE_INTERRUPTS_S;

             /* DISABLE 8274 INTERRUPTS - SET UP THE 8259A */

67   1       DISABLE_INTERRUPTS_S: PROCEDURE PUBLIC;

68   2       DISABLE;

69   2       INT_VEC = INT_VEC_STORE;

70   2       OUTPUT(OCW1_59) = MASK_59;
71   2       ENABLE;

72   2       RETURN;
73   2       END DISABLE_INTERRUPTS_S;

             /* CHANNEL B RECEIVE CHARACTER AVAILABLE */

74   1       CHB_RX_CHAR: PROCEDURE;

75   2       TEMP=INPUT(DATA_B_74);

76   2       OUTPUT(COMMAND_B_74) = 38H;
77   2       RETURN;
78   2       END CHB_RX_CHAR;

             $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST


             /***************************/
             /* MAIN INTERRUPT ROUTINE */
             /***************************/

79   1       INT_8274_S: PROCEDURE INTERRUPT 35 PUBLIC;

80   2       OUTPUT(COMMAND_B_74) = 2;                /* SET POINTER TO 2*/
81   2       TEMP = INPUT(STATUS_B_74) AND 07H;       /* READ INTERRUPT VECTOR */
                                                      /* CHECK FOR CHA INT ONLY*/
             /* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
82   2       DO CASE TEMP;
83   3           CALL    IGNORE_INT;                  /* V2V1V0 = 000*/
84   3           CALL    IGNORE_INT;                  /* V2V1V0 = 001*/
85   3           CALL    CHB_RX_CHAR;                 /* V2V1V0 = 010*/
86   3           CALL    IGNORE_INT;                  /* V2V1V0 = 011*/
87   3           CALL    IGNORE_INT;                  /* V2V1V0 = 100*/
88   3           CALL    CHA_EXTERNAL_CHANGE;         /* V2V1V0 = 101*/
89   3           CALL    CHA_RX_CHAR;                 /* V2V1V0 = 110*/
90   3           CALL    CHA_RX_SPECIAL;              /* V2V1V0 = 111*/
91   3       END;
92   2       OUTPUT(COMMAND_A_74) =38H;   /* END OF INTERRUPT FOR 8274 */
93   2       OUTPUT(OCW2_59) = 63H;       /* 8259 EOI */
94   2       OUTPUT(OCW1_59) = INPUT(OCW1_59) AND 0F7H;
95   2       RETURN;
96   2       END INT_8274_S;



             /* DEFAULT INTERRUPT ROUTINE - 8259A INTERRUPT 7 */
             /* REQUIRED ONLY WHEN DMA CONTROLLER IS ENABLED  */
             /* BEFORE RECEIVING FIRST CHARACTER WHICH IS     */
             /* AT HIGH BAUD RATES  LIKE 800K BAUD. READ APP. */
             /* NOTE SECTION 6 FOR DETAILS                    */
```

210403-21

```
 97   1        INT_39: PROCEDURE INTERRUPT 39;
 98   2                OUTPUT(OCW2_59) = 20H;     /* NON-SPECIFIC EOI */
 99   2                OUTPUT(OCW1_59) = INPUT(OCW1_59) AND OF7H;
100   2                RESULTS_S = FAIL;
101   2        END INT_39;

102   1        END INTR_8274_S;


MODULE INFORMATION:

     CODE AREA SIZE       = 01BFH     447D
     CONSTANT AREA SIZE   = 0000H       0D
     VARIABLE AREA SIZE   = 0006H       6D
     MAXIMUM STACK SIZE   = 0022H      34D
     295 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE STEST
OBJECT MODULE PLACED IN :F1:STEST.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:STEST.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)
COMPACT NOINTVECTOR ROM

              /***********************************************************************/
              /*                                                                     */
              /*       iSBC 545 PORT A (8274) SDLC TEST                              */
              /*                                                                     */
              /***********************************************************************/

  1           STEST: DO;


  2   1       DELAY_S: PROCEDURE EXTERNAL;
  3   2       END DELAY_S;

  4   1       ENABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  5   2       END ENABLE_INTERRUPTS_S;

  6   1       DISABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  7   2       END DISABLE_INTERRUPTS_S;

  8   1       INIT_8274_SDLC_S: PROCEDURE EXTERNAL;
  9   2       END INIT_8274_SDLC_S;

 10   1       INIT_8237_S: PROCEDURE EXTERNAL;
 11   2       END INIT_8237_S;

 12   1       STOP_8237_S: PROCEDURE EXTERNAL;
 13   2       END STOP_8237_S;

 14   1       VERIFY_TRANSFER_S: PROCEDURE EXTERNAL;
 15   2       END VERIFY_TRANSFER_S;

 16   1       INT_8274_S: PROCEDURE INTERRUPT 35 EXTERNAL;
 17   2       END INT_8274_S;
              $NOLIST
              $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


 28   1       DECLARE (RESULTS_S,TXDONE_S,RXDONE_S) BYTE PUBLIC;
 29   1       DECLARE DONE            LIT      'OFFH',
                      NOT_DONE        LIT      '00H',
                      PASS            LIT      'OFFH',
                      FAIL            LIT      '00H';
```

210403-22

```
            $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

  30   1       CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

  31   2               CALL     ENABLE_INTERRUPTS_S;
  32   2               CALL     INIT_8274_SDLC_S;
  33   2               ENABLE;
  34   2               OUTPUT(COMMAND_A_74) = 28H;   /* RESET TX INT/DMA        */
  35   2               OUTPUT(COMMAND_B_74) = 28H;   /* BEFORE INITIALIZING 8237*/
  36   2               CALL     INIT_8237_S;
  37   2               OUTPUT(DATA_A_74) = 55H;   /* LOAD FIRST CHARACTER FROM CPU*/

                       /* TO ENSURE CRC TRANSMISSION RESET TX UNDERRUN LATCH*/
  38   2               OUTPUT(COMMAND_A_74) = 0C0H;
  39   2               RXDONE_S,TXDONE_S=NOT_DONE;          /* CLEAR ALL FLAGS */
  40   2               RESULTS_S=PASS;                      /* FLAG SET FOR MONITOR*/

  41   2               DO WHILE TXDONE_S=NOT_DONE;          /* DO UNTIL TERMINAL COUNT*/
  42   3               END;

  43   2               DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
                       /* WAIT FOR CRC TO GET TRANSMITTED */
                       /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
  44   3               END;
  45   2               DO WHILE RXDONE_S=NOT_DONE;          /* DO UNTIL TERMINAL COUNT*/
  46   3               END;

  47   2               CALL     STOP_8237_S;

  48   2               CALL     DISABLE_INTERRUPTS_S;

  49   2               CALL     VERIFY_TRANSFER_S;

  50   2               RETURN RESULTS_S;

  51   2       END CHA_SDLC_TEST;
  52   1       END STEST;


MODULE INFORMATION:

     CODE AREA SIZE     = 0063H     99D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0004H      4D
     198 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION


PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE VECTOR_MODE
OBJECT MODULE PLACED IN :F1:VECTOR.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:VECTOR.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)


               /*********************************************************************/
               /*                                                                   */
               /*            8274 INTERRUPT HANDLING ROUTINE FOR                    */
               /*                    8274 VECTOR MODE                               */
               /*                 STATUS AFFECTS VECTOR                             */
               /*                                                                   */
               /*********************************************************************/
```

210403-23

```
          /* THIS IS AN EXAMPLE OF HOW 8274 CAN BE USED IN VECTORED MODE.        */
          /* THE iSBC88/45 BOARD WAS REWIRED TO DISABLE THE PIT 8259A AND        */
          /* ENABLE THE 8274 TO PLACE ITS VECTOR ON THE DATABUS IN RESPONSE      */
          /* TO THE INTA SEQUENCE FROM THE 8088. OTHER MODIFICATIONS INCLUDED */
          /* CHANGES TO 8274 INITIALIZATION PROGRAM (SINI74) TO PROGRAM 8274    */
          /* INTO VECTORED MODE (WRITE REGISTER 2A D5=1).                        */

  1       VECTOR_MODE: DO;
          $NOLIST

 12   1   DECLARE TEMP BYTE;
 13   1   DECLARE (RESULTS_S, TXDONE, RXDONE) BYTE EXTERNAL;
 14   1   DECLARE DONE  LITERALLY 'OFFH',
                  NOT_DONE LITERALLY 'OOH',
                  PASS     LITERALLY 'OFFH',
                  FAIL     LITERALLY 'OOH';

          /*******************************************************************/
          /*      TRANSMIT INTERRUPT CHANNEL A INTERRUPT WILL NOT BE SEEN IN THE */
          /*      DMA OPERATION.                                               */
          /*******************************************************************/

 15   1       TX_INTERRUPT_CHA: PROCEDURE INTERRUPT 84;
 16   2       OUTPUT(COMMAND_A_74) = 00101000B;          /*RESET TXINT PENDING*/
 17   2       OUTPUT(COMMAND_A_74) = 00111000B;          /*EOI*/
 18   2       END TX_INTERRUPT_CHA;

          /*******************************************************************/
          /*      EXTERNAL/STATUS INTERRUPT PROCEDURE: CHECKS FOR END OF MESSAGE */
          /*      ONLY. IF THIS IS NOT TRUE THEN THE FAIL FLAG IS SET. HOWEVER, */
          /*      A USER PROGRAM SHOULD CHECK FOR OTHER EXT/STATUS CONDITIONS    */
          /*      ALSO IN RR1 AND THEN TAKE APPROPRIATE ACTION BASED ON THE      */
          /*      APPLICATION.                                                 */
          /*******************************************************************/

 19   1   EXT_STAT_CHANGE_CHA: PROCEDURE INTERRUPT 85;
 20   2           TEMP  = INPUT(STATUS_A_74);
 21   2           IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
 22   2              TXDONE = DONE;
 23   2           ELSE DO;
 24   3              TXDONE = DONE;

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

 25   3                 RESULTS_S = FAIL;
 26   3              END;

 27   2        OUTPUT(COMMAND_A_74) = 00010000B;           /*RESET EXT STAT INT*/
 28   2              OUTPUT(COMMAND_A_74) = 00111000B;          /*EOI*/
 29   2              RETURN;
 30   2   END EXT_STAT_CHANGE_CHA;

          /*******************************************************************/
          /*     RECEIVER CHARACTER AVAILABLE INTERRUPT WILL APPEAR ONLY ON FIRST*/
          /*     RECEIVE CHARACTER. SINCE DMA CONTROLLER HAS BEEN ENABLED BEFORE */
          /*     THE FIRST CHARACTER IS RECEIVED, THE RECEIVER REQUEST IS        */
          /*     SERVICED BY THE DMA CONTROLLER.                                 */
          /*******************************************************************/

 31   1   RX_CHAR_AVAILABLE_CHA: PROCEDURE INTERRUPT 86;
 32   2       OUTPUT(COMMAND_A_74) = 00111000B;         /*EOI*/
 33   2           RETURN;
 34   2   END RX_CHAR_AVAILABLE_CHA;
          $EJECT
```

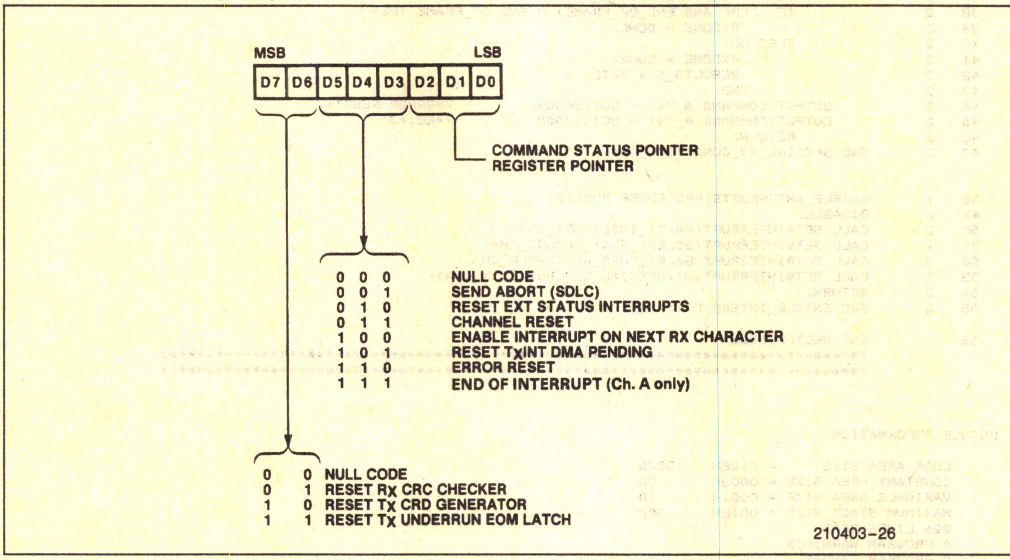
210403-24

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


                  /************************************************************************/
                  /*     SPECIAL RECEIVE CONDITION INTERRUPT SERVICE ROUTINE CHECKS FOR */
                  /*     END OF FRAME BIT ONLY. SEE SPECIAL SERVICE ROUTINE FOR NON-     */
                  /*     VECTORED MODE FOR CRC CHECK AND OVERRUN ERROR CHECK.            */
                  /************************************************************************/

  35   1          SPECIAL_RX_CONDITION_CHA: PROCEDURE INTERRUPT 87;

  36   2                  OUTPUT(COMMAND_A_74) = 1;                   /*POINTER 1*/
  37   2                  TEMP = INPUT(STATUS_A_74);
  38   2                  IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
  39   2                        RXDONE = DONE;
  40   2                  ELSE DO;
  41   3                        RXDONE = DONE;
  42   3                        RESULTS_S = FAIL;
  43   3                       END;
  44   2              OUTPUT(COMMAND_A_74) = 00110000B;           /*ERROR RESET*/
  45   2              OUTPUT(COMMAND_A_74) = 00111000B;           /*EOI*/
  46   2                  RETURN;
  47   2          END SPECIAL_RX_CONDITION_CHA;


  48   1          ENABLE_INTERRUPTS: PROCEDURE PUBLIC;
  49   2          DISABLE;
  50   2          CALL SET$INTERRUPT(84, TX_INTERRUPT_CHA);
  51   2          CALL SET$INTERRUPT(85, EXT_STAT_CHANGE_CHA);
  52   2          CALL SET$INTERRUPT(86, RX_CHAR_AVAILABLE_CHA);
  53   2          CALL SET$INTERRUPT(87, SPECIAL_RX_CONDITION_CHA);
  54   2          RETURN;
  55   2          END ENABLE_INTERRUPTS;

  56   1          END VECTOR_MODE;
                  /****************************************************************************/
                  /****************************************************************************/


MODULE INFORMATION:

     CODE AREA SIZE     = 012EH    302D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 001EH     30D
     226 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

210403-25

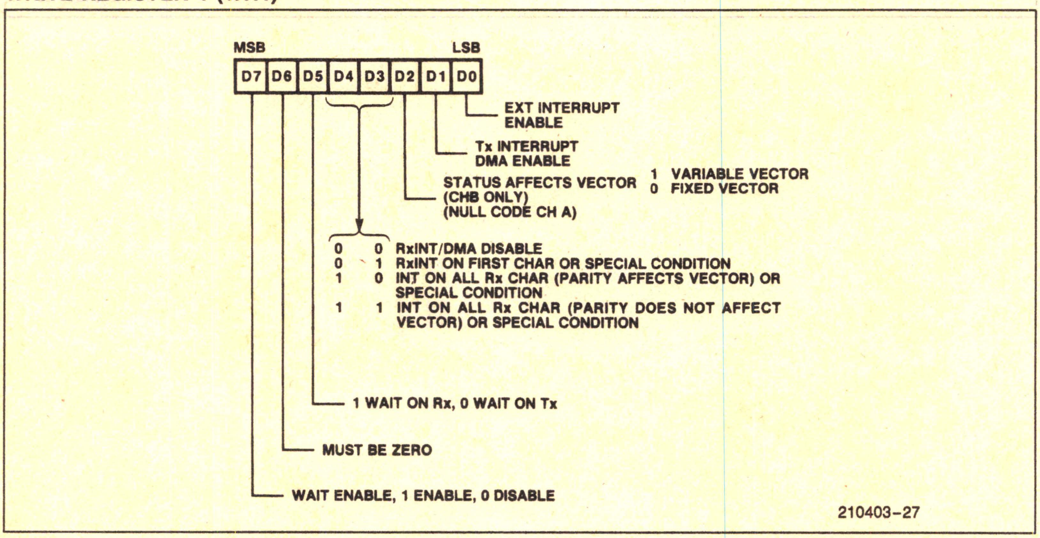
APPENDIX B
MPSC READ/WRITE REGISTER DESCRIPTIONS

WRITE REGISTER 0 (WR0)



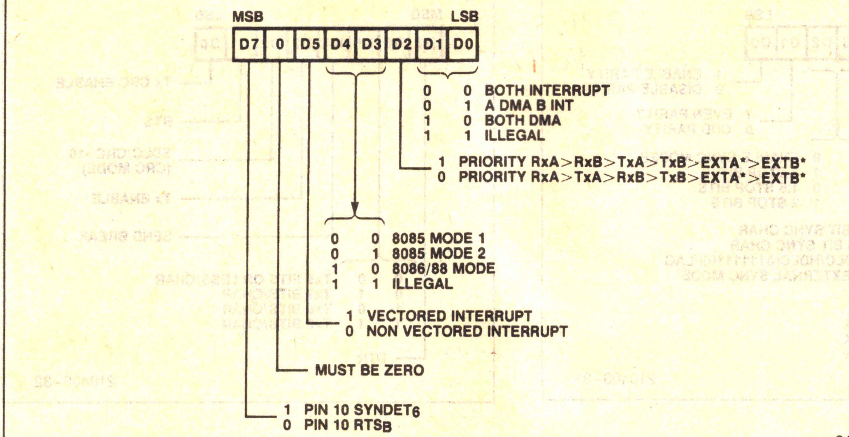
210403-26

WRITE REGISTER 1 (WR1)



210403-27

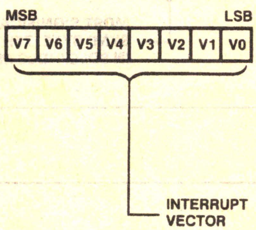
WRITE REGISTER 2 (WR2): CHANNEL A



210403-28

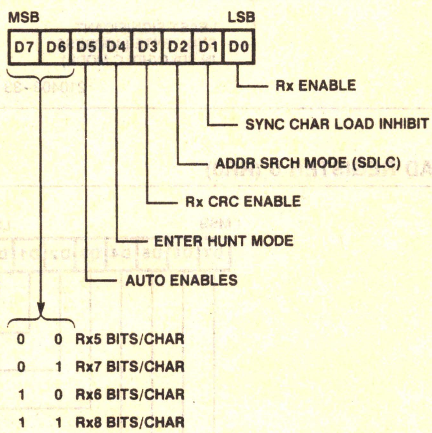
*External Status Interrupt only if EXT Interrupt Enable (WR1: D0) is set.

WRITE REGISTER 2 (WR2): CHANNEL B



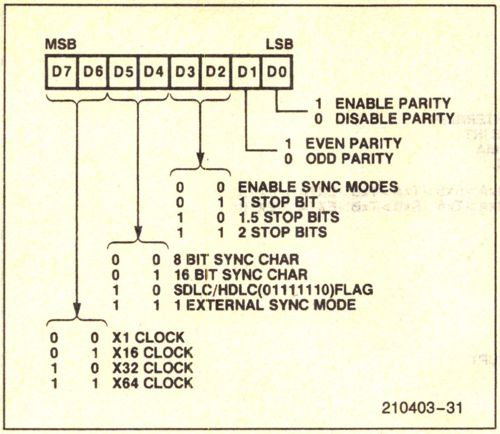
210403-29

WRITE REGISTER 3 (WR3)



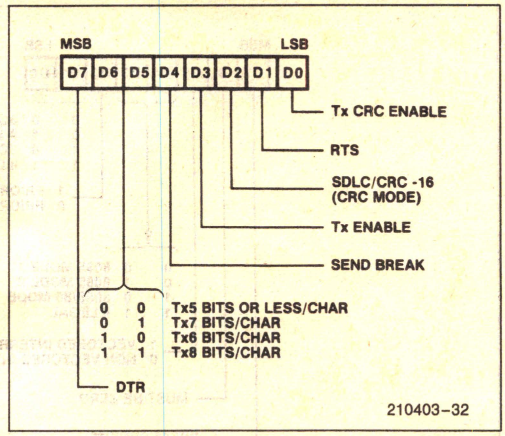
210403-30

WRITE REGISTER 4 (WR4)



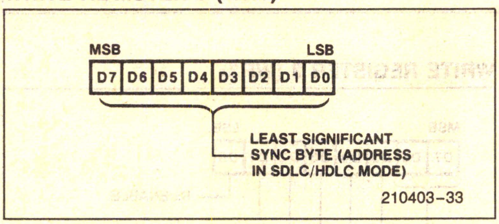
210403-31

WRITE REGISTER 5 (WR5)



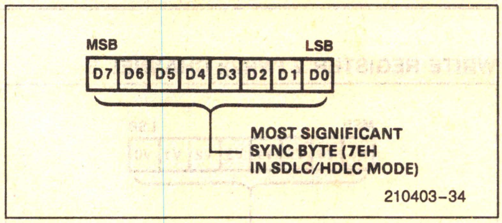
210403-32

WRITE REGISTER 6 (WR6)



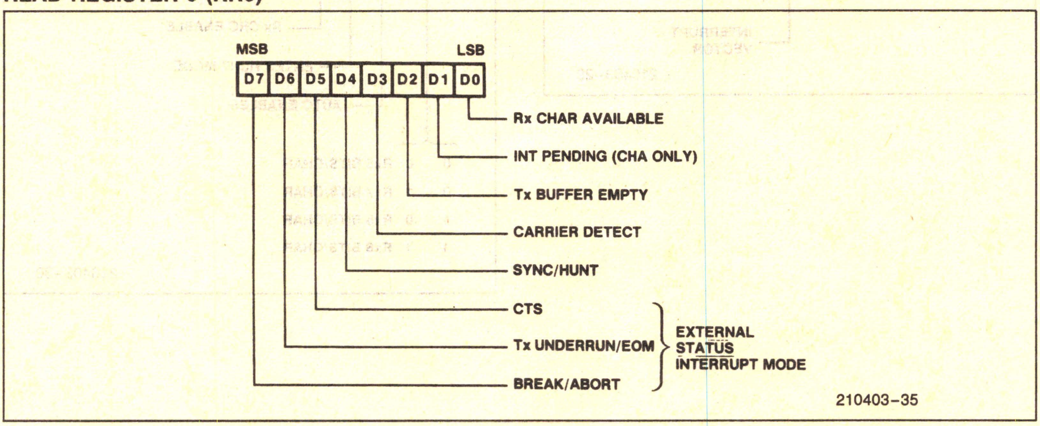
210403-33

WRITE REGISTER (WR7)



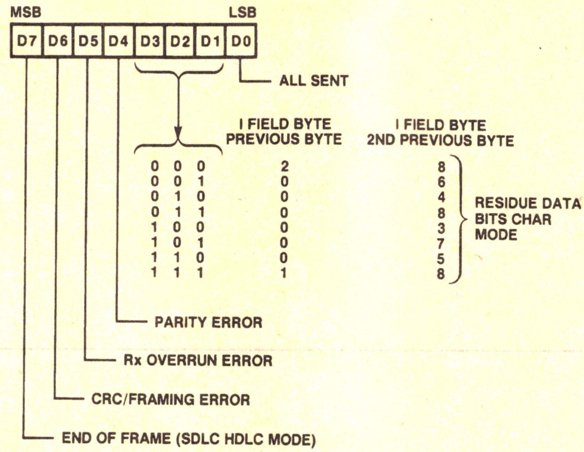
210403-34

READ REGISTER 0 (RR0)



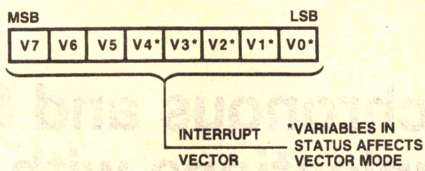
210403-35

READ REGISTER 1 (RR1): (SPECIAL RECEIVE CONDITION MODE)



210403-36

READ REGISTER 2 (RR2) CHANNEL B ONLY



210403-37

REFERENCES

1. *IBM Document No. GA27-3004-2: General Information—Binary Synchronous Communications*
2. *Application Note AP134: Asynchronous Communication with the 8274 Multiple Protocol Serial Controller.* Intel Corp., Ca.
3. *8274 MPSC Data Sheet,* Intel Corporation, Ca.
4. *iSBC 88/45 Hardware Reference Manual,* Intel Corp., Ca.
5. *Computer Networks and Distributed Processing by James Martin.* Prentice Hall, Inc., N.J.

intel®

APPLICATION NOTE

AP-222

November 1986

Asynchronous and SDLC Communications with 82530

SHARAD GANDHI

Order Number: 231262-002

INTRODUCTION

INTEL's 82530, Serial Communications Controller (SCC), is a dual channel, multi-protocol data communications peripheral. It is designed to interface to high speed communications lines using asynchronous, byte synchronous, and bit synchronous protocols. It runs up to 1.5 Mbits/sec, has on-chip baud rate generators and on-chip NRZI encoding and decoding circuits—very useful for SDLC communication. This application note shows how to write I/O drivers for the 82530 to do initialization and data links using asynchronous (ASYNC) and SDLC protocols. The appendix includes sections to show how the on-chip baud rate generators could be programmed, how the modem control pins could be used, and how the 82530 could be interfaced to INTEL's 80186/188 processors.

This article deals with the software for the following:

1. SCC port definition
2. Accessing the SCC registers
3. Initialization for ASYNC communication
4. ASYNC communication in polling mode
5. ASYNC communication in interrupt mode
6. Initialization for SDLC communication
7. SDLC frame reception
8. SDLC frame transmission
9. SDLC interrupt routines

The description is written around illustrations of the actual software written in PLM86 for a 80186 - 82530 system.

I. SCC Port Definition

The Figure 1 shows how the 4 ports (2 per channel) of the SCC can be defined. Note that the sequence of ports in the ascending order of addresses is *not* the one that is normally expected. In the ascending order it is: command (B), data (B), command (A) and data (A). In an 80186 - 82530 system, the interconnection is as follows:

	PCSn	—	CS	
	A1	—	D/C	
80186 pins	A2	—	A/B	82530 pins
	RD	—	RD	
	WR	—	WR	

2. Accessing the SCC Registers

The SCC has 16 registers on each of the channels (A and B). For each channel there is only one port, the command port, to access all the registers. The register #0 can be always accessed directly through the command port. All other registers are accessed indirectly through register #0. First, the number of the register to be accessed is written to the register #0 - see the statement, in Figure 2: 'output (ch_a_command) = reg_no and 0fh'. Then, the desired register is written to or read out. The Figure 2 shows 4 procedures: rra and wra, for reading and writing channel A registers; rrb and wrb, for reading and writing channel B registers. The read procedures are of the type 'byte' - they return the contents of the register being read. The write procedures require two parameters - the register number and the value to be written.

```
/*------------------------------------------------------------------------------*/

declare ch_b_command   literally 'pcs5 + 0', /* scc channel_b command word*/
        ch_b_data      literally 'pcs5 + 2', /* scc channel_b data word */
        ch_a_command   literally 'pcs5 + 4', /* scc channel_a command word */
        ch_a_data      literally 'pcs5 + 6'; /* scc channel_a data word */

/*------------------------------------------------------------------------------*/
```
231262-1

Figure 1. SCC Port Definition

```
/*---------------------------------------------------------------------------*/

/* read selected scc register */

rra: procedure (reg_no) byte;
        declare reg_no byte;

        if (reg_no and 0fh) <> 0
        then output(ch_a_command) = reg_no and 0fh;
        return input(ch_a_command);
end rra;

rrb: procedure (reg_no) byte;
        declare reg_no byte;

        if (reg_no and 0fh) <> 0
        then output (ch_b_command) = reg_no and 0fh;
        return input(ch_b_command);
end rrb;

/* write selected scc register */

wra: procedure (reg_no, value);
        declare reg_no byte;
        declare value byte;

        if (reg_no and 0fh) <> 0
        then output (ch_a_command) = reg_no and 0fh;
        output (ch_a_command) = value;
end wra;

wrb: procedure (reg_no, value);
        declare reg_no byte;
        declare value byte;

        if (reg_no and 0fh) <> 0
        then output (ch_b_command) = reg_no and 0fh;
        output (ch_b_command) = value;
end wrb;

/*---------------------------------------------------------------------------*/
```

231262-2

Figure 2. Accessing the SCC Registers

3. Initialization for ASYNC Operation

In the following example, channel B of the SCC is used to perform ASYNC communication. Figure 3 shows how the channel B is initialized and configured for ASYNC operation. This is done by writing the various channel B registers with the proper parameters as shown. The comments in the program show what is achieved by each statement. After a software reset of the channel, register #4 should be written before writing to the other registers. The on-chip Baud Rate Generator is used to generate a 1200 bits/sec clock for both the transmitter and the receiver. The interrupts for transmitter and/or receiver are enabled only for the interrupt mode of operation; for polling, interrupts must be kept disabled.

4. ASYNC Communication in Polling Mode

Figure 4 shows the procedures for reading in a received character from the 82530 (scc_in) and for writing out a character to the 82530 (scc_out) in the polling mode.

The scc_in procedure returns a byte value which is the character read in. The receiver is polled to find if a character has been received by the SCC. Only when a character has been received, the character is read in from the data port of the SCC channel B.

The scc_out procedure requires a byte parameter which is the character being written out. The transmit-

```
/*----------------------------------------------------------------------------*/

scc_init_b: procedure;

/* scc ch B register initialization for ASYNC mode */

      call wrb(09, 01000000b);       /* channel B reset */
      call wrb(04, 11001110b);       /* 2 stop, no parity, brf = 64x */
      call wrb(02, 00100000b);       /* vector = 20h */
      call wrb(03, 11000000b);       /* rx 8 bits/char, no auto-enable */
      call wrb(05, 01100000b);       /* tx 8 bits/char */
      call wrb(06, 00000000b);
      call wrb(07, 00000000b);
      call wrb(09, 00000001b);       /* vector includes status */
      call wrb(10, 00000000b);
      call wrb(11, 01010110b);       /* rxc = txc = BRG , trxc = BRG out */
      call wrb(12, 00011000b);       /* to generate 1200 baud, x64 @ 4 mhz */
      call wrb(13, 00000000b);
      call wrb(14, 00000011b);       /* BRG source = SYS CLK, enable BRG */
      call wrb(15, 00000000b);       /* all ext status interrupts off */

  /* enables */

      call wrb(03, 11000001b);        /* scc-b receive enable */
      call wrb(05, 11101010b);        /* scc-b transmit enable, dtr on, rts on */

    /* enable interrupts - only for interrupt driven ASYNC I/O */

      call wrb(09, 00001001b);       /* master IE, vector includes status */
      call wrb(01, 00010011b);       /* tx ,rx, ext interrupts enable */

end scc_init_b;

/*----------------------------------------------------------------------------*/
```

231262-3

Figure 3. Initialization for ASYNC Communication

```
/*-------------------------------------------------------------------------------------*/

/* scc data character input from channel B */

scc_in: procedure byte;

      declare char byte;

      do while (input(ch_b_command) and 1h) = 0; end;
      char = input(ch_b_data);     /* if rx data character is available */
      return char;                 /* then input it to buffer */

end scc_in;

/* scc data character output to channel B */

scc_out: procedure (char);

      declare char byte;

      do while (input(ch_b_command) and 4h) = 0; end;
      output(ch_b_data) = char;    /* if tx buff empty then transfer the */
                                   /* data character to tx buff */

end scc_out;

/*-------------------------------------------------------------------------------------*/
```

231262-4

Figure 4. ASYNC Communication in Polling Mode

ter is polled for being ready to transmit the next character before writing the character out to the data port of SCC channel B.

Typical calls to these procedures are:

```
abc__variable = scc__in;
call scc__out (xyz__variable);
```

5. ASYNC Communication in Interrupt Mode

In contrast to polling for the receiver and/or the transmitter to be ready with/for the next character, the 82530 can be made to interrupt when it is ready to do receive or transmit.

The on-chip interrupt controller of the SCC can be made to operate in the vectored mode. In this mode, it generates interrupt vectors that are characteristic of the event causing the interrupt. For the example here, the vector base is programmed at 20h and 'Vector Includes Status' (VIS) mode is set - WR9 = XXX0XX01. Vectors and the associated events are:

Vector	Procedure	Event Causing Interrupt
20h	txintr__b	ch__b - transmit buffer empty
22h	esi__b	ch__b - external/status change
24h	rxintr__b	ch__b - receive character available
26h	src__b	ch__b - special receive condition
28h	txintr__a	ch__a - transmit buffer empty
2ah	esi__a	ch__a - external/status change
2ch	rxintr__a	ch__a - receive character available
2eh	src__a	ch__a - special receive condition

NOTE:
Odd vector numbers do not exist.

Figure 5 shows the interrupt procedures for the channel B operating in ASYNC mode. The transmitter buffer empty interrupt occurs when the transmitter can accept one more character to output. In the interrupt procedure for transmit, the byte char__out__530 is output. Following this, is an epiloge that is *common to all the*

interrupt procedures; the first statement is an end of interrupt command to the 82530 - *note* that it is issued to *channel A* - and the second is an End of Interrupt (EOI) command to the 80186 interrupt controller which is, in fact, receiving the interrupt from the 82530.

The receive buffer full interrupt occurs when the receiver has at least one character in its buffer, waiting to be read in by the CPU.

The esi__b is not enabled to occur and src__b cannot occur in the ASYNC mode unless the receiver is overrun or a parity error occurs.

```
/*---------------------------------------------------------------------------*/
/* channel B interrupt procedures */

txintr_b:     procedure     interrupt 20h;

        output (ch_b_data) = char_out_530;

        call wra(00,38h);                    /* reset highest IUS */
        output (eoir_186)  = 8000h;          /* non specific EOI */
        return;
end txintr_b;

esi_b:        procedure     interrupt 22h;

        call wrb(00,10h);                    /* reset ESI */

        call wra(00,38h);                    /* reset highest IUS */
        output (eoir_186)   = 8000h;         /* non specific EOI */
        return;
end esi_b;

rxintr_b:     procedure     interrupt 24h;

        char_in_530 = input (ch_b_data);

        call wra(00,38h);                    /* reset highest IUS */
        output (eoir_186)   = 8000h;         /* non specific EOI */
        return;
end rxintr_b;

src_b:        procedure     interrupt 26h;

        call wrb(00,30h);                    /* error reset */

        call wra(00,38h);                    /* reset highest IUS */
        output (eoir_186)   = 8000h;         /* non specific EOI */
        return;
end src_b;

/*---------------------------------------------------------------------------*/
```

231262-5

Figure 5. ASYNC Communication in Interrupt Mode

6. Initialization for SDLC Communication

Channel A of the SCC is programmed for being used for SDLC operation. It uses the DMA channels on the 80186. Figure 6 shows the initialization procedure for channel A. The comments in the software show the effect of each statement. The on-chip Baud Rate Generator is used to generate a clock of 125 kHz both for reception and transmission. This procedure is just to prepare the channel A for SDLC operation. The actual transmission and reception of frames is done using the procedures described further.

7. SDLC Frame Reception

Figure 7 shows the entire set-up necessary to receive a SDLC frame. First the DMA controller is programmed with the receive buffer address (@rx__buff), byte count, mode etc and is also enabled. Then a flag indicating reception of the frame is reset. An Error Reset command is issued to clear up any pending error conditions. The receive interrupt is enabled to occur at the end of frame reception (Special Receive Condition); lastly, the receiver is enabled and put in the Hunt mode (to detect the SDLC flag). When the first flag is detected on the RxDA pin, it goes from the Hunt to the Sync mode. It receives the frame and the end of frame interrupt (src__b, vector = 2eh) occurs.

8. SDLC Frame Transmission

Figure 8 shows the procedure for transmitting a SDLC frame once channel A is initialized. The DMA controller is initialized with the transmit buffer address (@tx__buff (1)) - *note*, it is the second byte of the transmit buffer - and the byte count - again one less than the total buffer length. This is done because the first byte in the buffer is output directly using an I/O instruction and not by DMA. Then the flag indicating frame transmitted is reset. The events following are very critical in sequence:

a. Reset external status interrupts

b. Enable the transmitter

c. Reset transmit CRC

d. Enable transmitter underrun interrupt

e. Enable the DMA controller

f. Output first byte of the transmit block to data port

g. Reset Transmit Underrun Latch

```
/*---------------------------------------------------------------------------*/

scc_init_a: procedure;

/* scc ch A register initialization for SDLC mode */

      call wra(09, 10000000b);        /* channel A reset */
      call wra(04, 00100000b);        /* SDLC mode */
      call wra(01, 01100000b);        /* DMA for Rx */
      call wra(03, 11000000b);        /* 8 bit Rx char, Rx disable */
      call wra(05, 01100000b);        /* 8 bit Tx char, Tx disable */
      call wra(06, 01010101b);        /* node address */
      call wra(07, 01111110b);        /* SDLC flag */
      call wra(10, 10000000b);        /* preset CRC, NRZ encoding */
      call wra(11, 01010110b);        /* rxc = txc = BRG , trxc = BRG out */
      call wra(12, 00001110b);        /* to generate 125 Kbaud, x1 @ 4 mhz */
      call wra(13, 00000000b);
      call wra(14, 00000110b);        /* BRG source = SYS CLK, DMA for Tx */
      call wra(15, 00000000b);        /* all ext status interrupts off */

    /* enables */

      call wra(14, 00000111b);        /* enable : BRG */
      call wra(01, 11100000b);        /* enable : dreq */
      call wra(09, 00001001b);        /* master IE, vector includes status */

end scc_init_a;

/*---------------------------------------------------------------------------*/
```

231262-6

Figure 6. Initialization for SDLC Communication

```
/*---------------------------------------------------------------------------*/

rx_init: procedure;

     declare dma_0_mode literally '1010001001000000b';
     /* src=IO, dest=M(inc), sync=src, TC, noint, priority, byte */

     outword(dma_0_dpl) = low16(@rx_buff);
     outword(dma_0_dph) = high16(@rx_buff);
     outword(dma_0_spl) = ch_a_data;
     outword(dma_0_sph) = 0;
     outword(dma_0_tc)  = block_length + 2;       /* +2 for CRC */
     outword(dma_0_cw)  = dma_0_mode or 0006h;    /* start DMA channel 0 */

     frame_recd = 0;                   /* reset frame received flag */

     call wra(00,30h);                 /* error reset */
     call wra(01, 11111001b);          /* sp. cond intr only, ext int enable */
     call wra(03, 11010001b);          /* enable receiver, enter hunt mode */

end rx_init;

/*---------------------------------------------------------------------------*/
```

231262-7

Figure 7. SDLC-DMA Frame Reception

```
/*---------------------------------------------------------------------------*/

tx_init: procedure;

     declare dma_1_mode literally '0001011010000000b';
     /* src=M(inc), dest=IO, sync=dest, TC, noint, noprior, byte */

     outword(dma_1_spl) = low16(@tx_buff(1));
     outword(dma_1_sph) = high16(@tx_buff(1));
     outword(dma_1_dpl) = ch_a_data;
     outword(dma_1_dph) = 0;
     outword(dma_1_tc)  = block_length - 1;     /* -1 for first byte */

     frame_tx = 0;                     /* reset frame transmitted flag */

     call wra(00, 00010000b);          /* reset ESI */
     call wra(05, 01101011b);          /* enable transmitter */
     call wra(00, 10101000b);          /* reset tx CRC, TxINT pending */
     call wra(15, 01000000b);          /* enable : TxU int */

     outword(dma_1_cw)  = dma_1_mode or 0006h;     /* start DMA channel 1 */
     output(ch_a_data) = tx_buff(0);       /* first byte - address field */
     call wra(00, 11000000b);              /* Reset Tx Underrun latch */

end tx_init;

/*---------------------------------------------------------------------------*/
```

231262-8

Figure 8. SDLC-DMA Frame Transmission

```
/*-------------------------------------------------------------------------*/

/* channel A interrupt procedures */

txintr_a:     procedure    interrupt 28h;

     call wra(00,38h);                  /* reset highest IUS */
     output (eoir_186)  = 8000h;        /* non specific EOI */
     return;
end txintr_a;

esi_a:        procedure    interrupt 2ah;

     call wra(00,10h);                  /* reset ESI */
     tx_stat = rra(0);                  /* read in status */
     frame_tx = 0ffh;                   /* set frame transmitted flag */

     call wra(00,38h);                  /* reset highest IUS */
     output (eoir_186)  = 8000h;        /* non specific EOI */
     return;
end esi_a;

rxintr_a:     procedure    interrupt 2ch;

     call wra(00,38h);                  /* reset highest IUS */
     output (eoir_186)  = 8000h;        /* non specific EOI */
     return;
end rxintr_a;

src_a:        procedure    interrupt 2eh;

     rx_stat = rra(1);
     call wra(00,30h);                  /* error reset */
     call wra(03,11000000b);            /* disable rx */
     frame_recd = 0ffh;                 /* set frame received flag */

     call wra(00,38h);                  /* reset highest IUS */
     output (eoir_186)  = 8000h;        /* non specific EOI */
     return;
end src_a;

/*-------------------------------------------------------------------------*/
```

231262-9

Figure 9. SDLC-DMA Interrupt Routines

The frame gets transmitted out with all bytes, except the first one, being fetched by the SCC using the DMA controller. At the end of the block the DMA controller stops supplying bytes to the SCC. This makes the transmitter underrun. Since the Transmitter Underrun Latch is in the reset state at this moment, the CRC bytes are appended by the SCC at the end of the transmit block going out. An External Status Change interrupt (esi__a, vector = 2ah) is generated with the bit for transmitter underrun set in RR0 register. This interrupt occurs when the CRC is being transmitted out and *not* when the frame is completely transmitted out.

9. SDLC Interrupt Routines

Figure 9 shows all the interrupt procedures for channel A when operating in the SDLC mode. The procedures of significance here are esi__a and src__a.

The end of frame reception results in the src__a procedure getting executed. Here the status in register RR1 is stored in a variable rx__stat for future examination. Any error bits set in status are reset, receiver is disabled and the flag indicating reception of a new frame is set.

The esi__a procedure is executed when CRC of the transmitted frame is just going out of the SCC. Reset External Status Interrupt command is executed, the external status is stored in a variable tx__stat for future examination and the flag indicating transmission of the frame is set.

End of frame processing is required after both of these interrupt procedures. It involves looking at rx__stat and tx__stat and checking if the desired operation was successful. The buffers used, may have to be recovered or new ones obtained to start another frame transmission or reception.

CONCLUSIONS

This article should ease the process of writing a complete data link driver for ASYNC and SDLC modes since most of the hardware dependent procedures are illustrated here. It was a conscious decision to make the procedures as small and easy to understand as possible. This had to be done at the expense of making the procedures general and not dealing with various exception conditions that can occur.

REFERENCES

1. 82530 Data Sheet, Order #230834-001
2. 82530 SCC Technical Manual, Order #230925-001

APPENDIX A
82530—BAUD RATE GENERATORS

The 82530 has two Baud Rate Generators (BRG) on chip—one for each channel. They are used to provide the baud rate or serial clock for receive and transmit operations. This article describes how the BRG can be programmed and used.

The BRG for each channel is totally independent of each other and have to be programmed separately for each channel. This article describes how any one of the two BRGs can be programmed for operation. To use the BRG, four steps have to be performed:

1. Determine the Baud Rate Time Constant (BRTC) to be programmed into registers WR12 (LSB) and WR13 (MSB).
2. Program in register WR11, to specify where the output of the BRG must go to.
3. Program the clock source to the BRG in register WR14.
4. Enable the BRG.

Step 1: Baud Rate Time Constant (BRTC)

The BRTC is determined by a simple formula:

$$\text{BRTC} = \frac{\text{Serial Clock Frequency}}{2 \times (\text{Baud Rate} \times \text{Baud Rate Factor})} - 2$$

Example:

For Serial Clock Frequency = 4 MHz

Baud Rate = 9600

Baud Rate Factor = 16

$$\text{BRTC} = \frac{4000000}{2 \times (9600 \times 16)} - 2$$

$$= 13.021 - 2 = 11.021$$

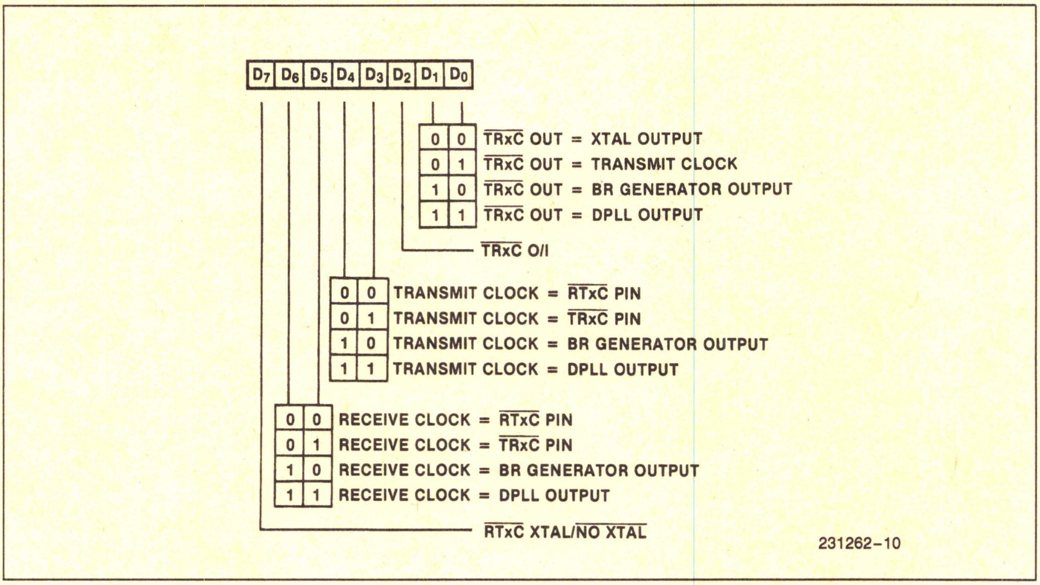


Figure 1. Write Register 11

Table 1. BRTC - Baud Rate Time Constant

		Baud Rate Factor			
		1	16	32	64
Baud Rate	9600	206.333	11.021	4.510	1.255
	4800	414.667	24.042	11.021	4.510
	2400	831.333	50.083	24.042	11.021
	1200	1664.667	102.167	50.083	24.042
	600	3331.333	206.333	102.167	50.083
	300	6664.667	414.667	206.333	102.167

Since only integers can be written into the registers WR12/WR13 this will have to be rounded off to 11 and it will result in an error of:

$$\frac{\text{fraction}}{\text{BRTC}} \times 100 = \frac{0.021}{11.021} \times 100 = 0.19\%$$

This error indicates that the baud rate signal generated by the BRG does not provide the exact frequency required by the system. This error is more serious for smaller baud rate factors. For asynchronous systems, errors up to 5% are considered acceptable.

Note that for BRTC = 0, BRG output frequency = 1/4 × Serial Clock Freq.

Table 1 shows the BRTC for a 4 MHz serial clock with various baud rates on the Y - axis and baud rate factors on the X - axis. The constant that is really programmed into registers WR12/WR13 is the integer closest to the BRTC value shown in the table.

Step 2: BRG Output

The output of the BRG can be directed to the Receiver, Transmitter and the TRxC output. This is programmed by setting bits D6 D5, bits D4 D3, and bits D1 D0 in register WR11 to 10. See Figure 1. The output of the BRG can also be directed to the Digital Phase Locked Loop (DPLL) for the on-chip decoding of the NRZI encoded received data signal. This is done by writing 100 into bits D7 D6 D5 of register WR14 as shown in Figure 2.

Step 3: BRG Source Clock

Register WR14 is used to select the input clock to the BRG. See Figure 2.

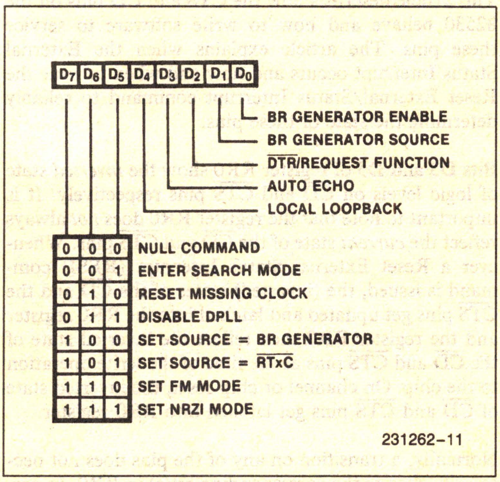


Figure 2. Write Register 14

WR14 / bit D1 = 0 → Clock comes from pin RTxC

WR14 / bit D1 = 1 → Clock comes from System Clock (PCLK)

On RESET WR14 / bit D1 = 0.

It should be noted that for the case of Bit D1 = 0, the clock comes either from:

a. Clock on pin RTxC - if WR11 / D7 = 0

or b. Crystal on pins RTxC & SYNC - if WR11 / D7 = 1

Step 4: BRG Enable

This is the last step where bit D0 of WR14 is set to start the BRG. The BRG can also be disabled by resetting this bit.

APPENDIX B
MODEM CONTROL PINS ON THE 82530

Introduction

This article describes how the $\overline{\text{CTS}}$ and $\overline{\text{CD}}$ pins on the 82530 behave and how to write software to service these pins. The article explains when the External Status Interrupt occurs and how and when to issue the Reset External/Status Interrupt command to reliably determine the state of these pins.

Bits D3 and D5 of register RR0 show the *inverted* state of logic levels on $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins respectively. It is important to note that the register RR0 does *not* always reflect the *current* state of the $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins. Whenever a Reset External/Status Interrupt (RESI) command is issued, the (inverted) states of the $\overline{\text{CD}}$ and the $\overline{\text{CTS}}$ pins get updated and latched into the RR0 register and the register RR0 then reflect the inverted state of the $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins at the time of the write operation to the chip. On channel or chip reset, the inverted state of $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins get latched into RR0 register.

Normally, a transition on any of the pins does not necessarily change the corresponding bit(s) in RR0. In certain situations it does and in some cases it does not. A sure way of knowing the current state of the pins is to read the register RR0 *after* a RESI command.

There are two cases:

I. External/Status Interrupt (ESI) enabled.

II. Polling (ESI disabled).

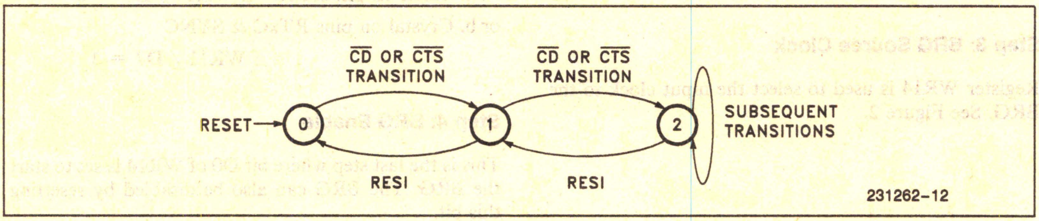
Case I: External Status Interrupt (ESI) Enabled

Whenever ESI is enabled, an interrupt can occur whenever there is a transition on $\overline{\text{CD}}$ or $\overline{\text{CTS}}$ pins - the IE bits for $\overline{\text{CD}}$ and/or $\overline{\text{CTS}}$ must also be set in WR15 for the interrupt to be enabled.

In this case, the first transition on any of these pins will cause an interrupt to occur and the corresponding bit in RR0 to change (even without the RESI command). A RESI command resets the interrupt line and also latches in the current state of both the $\overline{\text{CD}}$ and the $\overline{\text{CTS}}$ pins. If there was just one transition the RESI does not really change the contents of RR0.

If there are more than one transitions, either on the same pin or one each on both pins or multiple on both pins, the interrupt would get activated on the first transition and stay active. The bit in RR0 corresponding only to the very first transition is changed. All subsequent transitions have no effect on RR0. The first transition, in effect, freezes all changes in RR0. The first RESI command, as could be expected, latches the final (inverted) state of the $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins into the RR0 register. Note that all the intermediate transitions on the pins are lost (because the response to the interrupt was not fast enough). The interrupt line gets reset for only a brief moment following the first RESI command. This brief moment is approximately 500 ns for the 82530. After that the interrupt becomes active again. A second RESI command is necessary to reset the interrupt. Two RESI commands resets the interrupt line independent of the number of transitions occurred.

Whenever operating with ESI enabled, it is recommendable to issue two back-to-back RESI commands and then read the RR0 register to reliably determine the state of the $\overline{\text{CD}}$ and $\overline{\text{CTS}}$ pins and also to reset the interrupt line in case multiple transitions may have occurred.



State Diagram

Case II: Polling RR0 for $\overline{CD}$ and $\overline{CTS}$ Pins

If RR0 is polled for determining the state of the $\overline{CD}$ and $\overline{CTS}$ pins, then the External/Status Interrupt (ESI) is kept disabled. In this case the bits in RR0 may not change even for the first transition. The best way to handle this case to always issue a RESI command before reading in the RR0 register to determine the state of $\overline{CD}$ and $\overline{CTS}$ pins. Note, however, if two back-to-back RESI commands were to be issued every time before reading in the RR0 register, the first subsequent transition will change the corresponding bit in RR0.

The state diagram above illustrates how each transition on $\overline{CD}$ and $\overline{CTS}$ pins affect the 82530 and what effect the RESI command has.

State 0

It is entered on reset. No ESI due to $\overline{CTS}$ or $\overline{CD}$ are pending in this state. Any transition on $\overline{CTS}$ or $\overline{CD}$ pins lead to the state 1 *accompanied by an immediate change in the RR0 register.*

State 1

Interrupt is active (if enabled). If a RESI command is issued, state 0 is reached where interrupt is again inactive. However, a further transition on $\overline{CTS}$ or $\overline{CD}$ pin leads to state 2 *without an immediate change in RR0 register.*

State 2

Interrupt is active (if enabled). Any further transitions have no effect. A RESI command leads to state 1, temporarily making the interrupt inactive.

CONCLUSIONS

Register RR0 does not always reflect the current (inverted) state of the $\overline{CD}$ and $\overline{CTS}$ pins. The most reliable way to determine the state of the pins in interrupt or polling mode is to issue two back-to-back RESI commands and then read RR0. While polling, the second RESI is redundant but harmless. When issuing the back-to-back RESI commands to 82530 note that the separation between the two write cycles should be at least 6 CLK + 200 ns; otherwise the second RESI will be ignored.

APPENDIX C.
Interfacing 82530 to 80186

INTRODUCTION

The 82530 is Intel's new sophisticated dual channel multiprotocol serial communications controller. It can run up to 1.5 Mb/s in synchronous mode. It has useful features like on-chip baud rate generators and oscillators. It can be operated in polled, interrupt, half-duplex DMA, or full-duplex DMA modes. It is also capable of supplying its own interrupt vector during INTA cycles (like the 8274).

Interfacing the 82530 to the 8086/88 and 80186/188 processors requires the external logic shown in Figure 1.

FOUR TTL PACKAGE INTERFACE

A method of interfacing the 82530 to the 80186 CPU with four 14-pin TTL packages is described in this application note. The circuitry is shown in Figure 2. The TTLs are 74LS04, 74LS74, and 74LS08.

The interface supports the following operational modes:

1) Polled
2) Interrupt in vectored mode
3) Interrupt in non-vectored mode
4) Half-duplex - DMA on both channels
5) Full-duplex - DMA on one channel

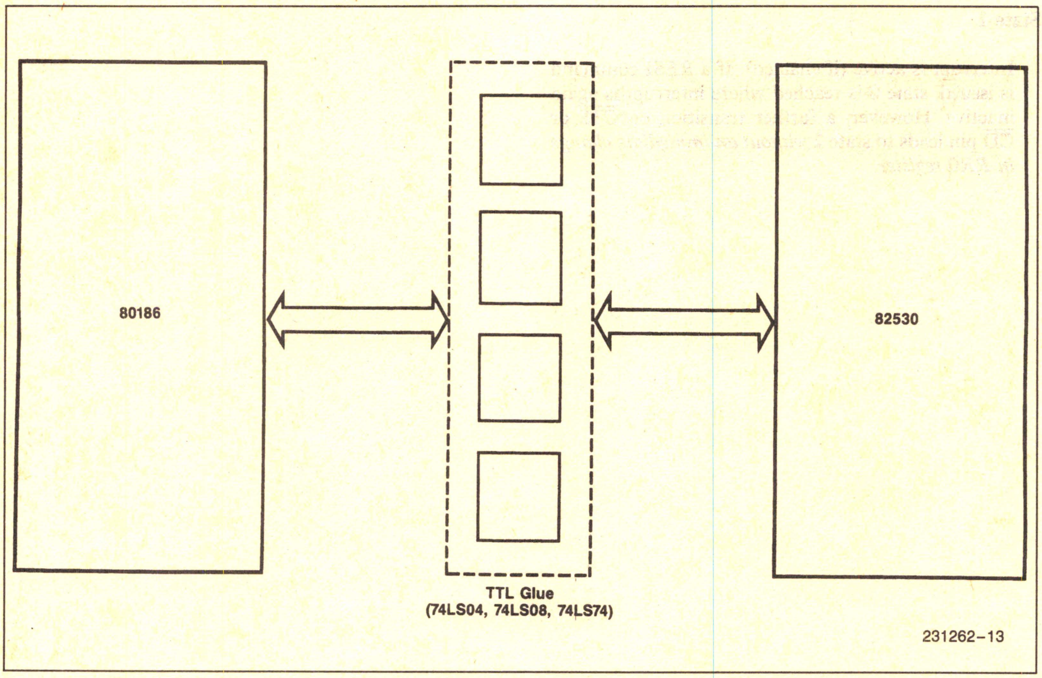


Figure 1. 80186/82530 Interface

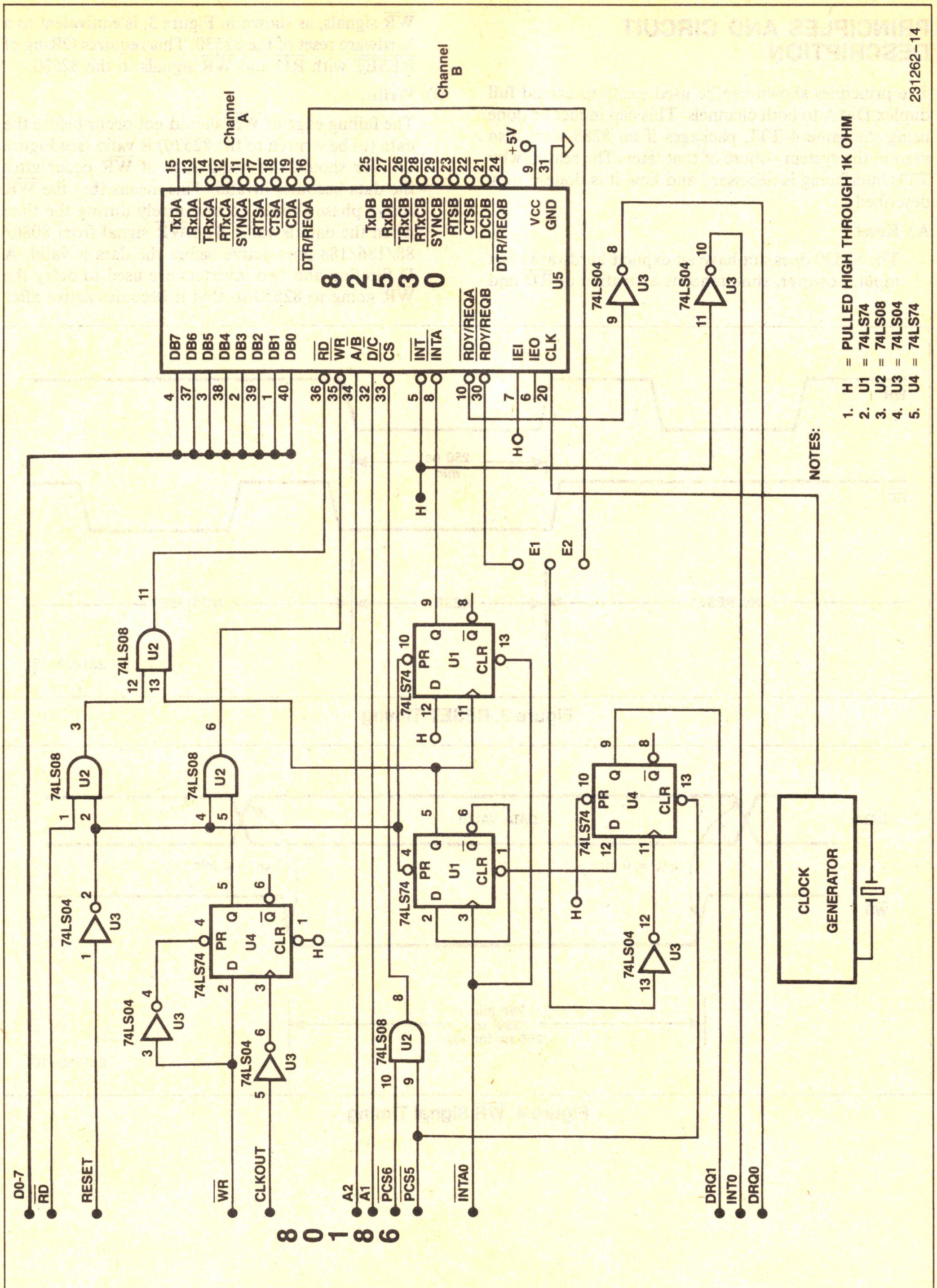


Figure 2. 4 TTL 82530 - 80186 Interface Circuit

PRINCIPLES AND CIRCUIT DESCRIPTION

The principles shown can be used easily to extend full duplex DMA to both channels. This can in fact be done using the same 4 TTL packages if an 8288 were also used in the system—more of that later. The reason why TTL interfacing is necessary and how it is done is now described.

A) **Reset**

The 82530 does not have an explicit hardware reset input; however, simultaneous activation of $\overline{RD}$ and $\overline{WR}$ signals, as shown in Figure 3, is equivalent to a hardware reset of the 82530. This requires ORing of $\overline{RESET}$ with $\overline{RD}$ and $\overline{WR}$ signals to the 82530.

B) **Write**

The falling edge of $\overline{WR}$ should not occur before the data (to be written to the 82530) is valid (see Figure 4). Nor should the rising edge of $\overline{WR}$ occur after the data becomes invalid. This means that the $\overline{WR}$ active phase should occur entirely during the time when the data is valid. The $\overline{WR}$ signal from 8086/88/186/188 goes active before the data is valid. A D flip-flop and two inverters are used to delay the $\overline{WR}$ going to 82530 so that it becomes active after

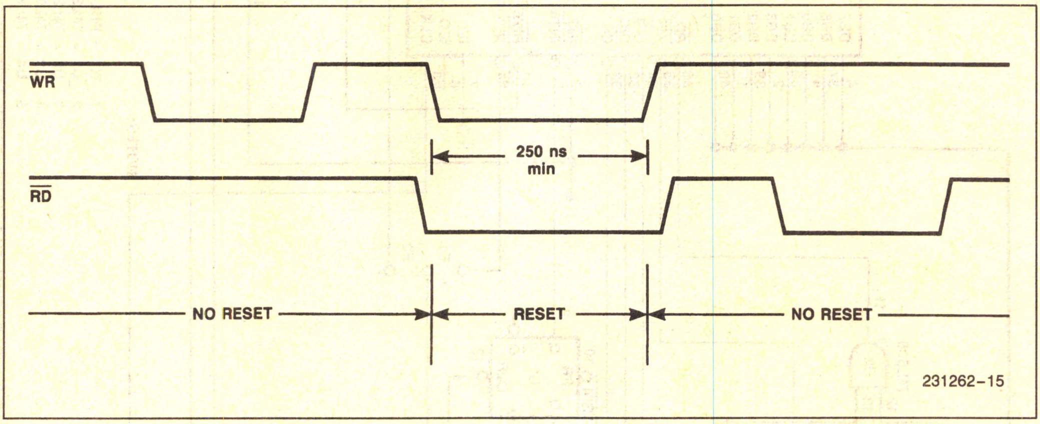


Figure 3. RESET Timing

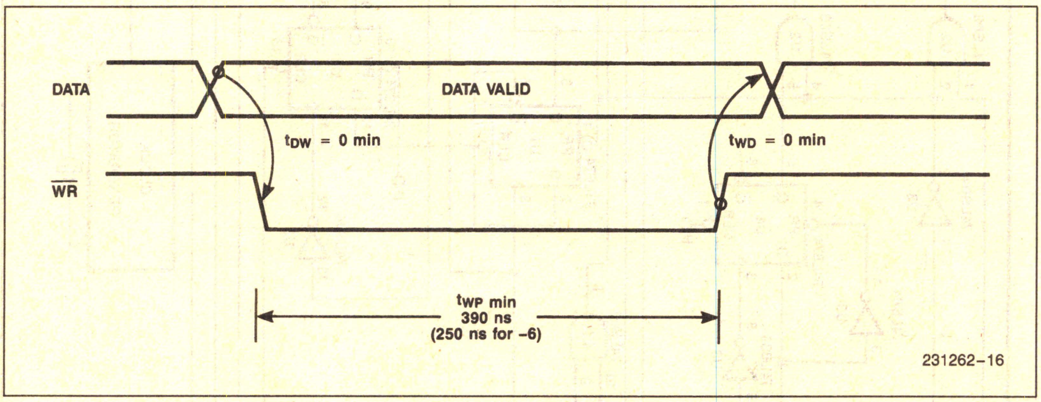


Figure 4. $\overline{WR}$ Signal Timing

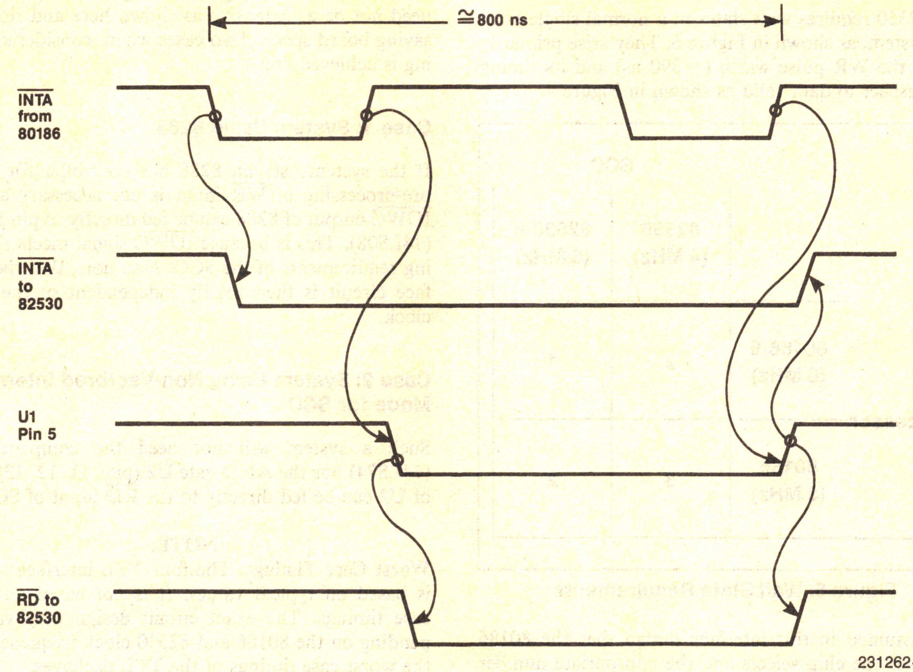


Figure 5. INTA Signal Processing

the data is valid. Note that if an 8288 is used to generate the $\overline{\text{IOWR}}$ signal (as in all big systems), then the flip-flop and inverters are not required since $\overline{\text{IOWR}}$ from the 8288 is compatible with the 82530 timing requirements.

C) **DMA**

The 82530 has two types of DMA request outputs; also, it has no DACK inputs. This means that the 82530 requires either a two cycle type of DMA transfer (a la 80186/88 or 8089), or DACK from the DMA controller (e.g. 8237A) has to be used to generate $\overline{\text{CS}}$, A/$\overline{\text{B}}$, and D/$\overline{\text{C}}$ signals.

The first type of DMA request is $\overline{\text{RDY}}/\overline{\text{REQ}}$. It can be programmed to function as RDY or $\overline{\text{DMAREQ}}$ (WR1: Bit 6). It can further be programmed as $\overline{\text{DMAREQ}}$ for transmit or for receive (WR1: Bit 5). This enables using just one signal for both the receive and transmit functions—ideal for half-duplex operation. This signal needs just an inversion to be fed into the DRQ input of the 80186.

The second DMA request signal is $\overline{\text{DTR}}/\overline{\text{REQ}}$. It can be programmed to function as $\overline{\text{DTR}}$ (Data Terminal Ready) or as $\overline{\text{DMAREQ}}$ for transmitter (active on transmitter buffer empty) in WR14: Bit 2. Thus, full-duplex DMA is possible by using $\overline{\text{DTR}}/\overline{\text{REQ}}$ as $\overline{\text{TxDRQ}}$ and $\overline{\text{RDY}}/\overline{\text{REQ}}$ as $\overline{\text{RxDRQ}}$. $\overline{\text{DTR}}/\overline{\text{REQ}}$ requires a little over 5 CLK cycles to become inactive. This would cause the DMA controller to run multiple DMA cycles, causing loss of data. A flip-flop is set by $\overline{\text{DTR}}/\overline{\text{REQ}}$ whose output is DRQ1 to the 80186. The response of the 80186 to DRQ1 is a read or write at $\overline{\text{PCS5}}$ address to do the DMA TRANSFER. This resets the flip-flop cutting off the DMA request to the 80186 which prevents false DMA transfer.

The DMA configurations supported by the interface are:

- Half-duplex on Channel A and Channel B
- Full-duplex on Channel A and no DMA on Channel B

D) **$\overline{\text{INTA}}$ Processing**

80186 generates 2 back-to-back $\overline{\text{INTA}}$ cycles in response to an interrupt and expects to read the interrupt vector on the second cycle. Two flip-flops (U1) are used to convert these two cycles to one $\overline{\text{INTA}}$ cycle and a $\overline{\text{RD}}$ pulse as required by the SCC. See timing diagram in Figure 5. SCC requires that the $\overline{\text{RD}}$ pulse is contained within the $\overline{\text{INTA}}$ pulse. This, along with the pulse width requirements for $\overline{\text{INTA}}$ and $\overline{\text{RD}}$ signals are easily met.

WAIT STATE REQUIREMENTS

The 82530 requires wait states in a normal single buffered system, as shown in Figure 6. They arise primarily due to the WR pulse width (=390 ns) and its timing with respect to data valid as shown in Figure 4.

		SCC 82530 (4 MHz)	SCC 82530-6 (6 MHz)
Processor	80186-6 (6 MHz)	2	1
Processor	80186 (8 MHz)	3	2

Figure 6. Wait State Requirements

It is assumed in this interface design that the 80186 generates the chip selects and the appropriate number of wait states. In an 8086/88 system, chip select and wait states must be generated externally just as for all other peripheral components attached to the CPU.

The $\overline{PCS6}$ chip select output from the 80186 is used to select the 82530 for all operations except to service DMA on Channel 1 of the 80186 when $\overline{PCS5}$ is used. Note that it is necessary to pulse $\overline{PCS5}$ signal before enabling the DMA Channel 1. This resets the DRQ1 flip-flop. A block for clock generator is also shown—although it is not considered a part of the CPU interface. It may be easily derived from CLKOUT.

The 4 TTL pack interface presented here covers all features of the SCC usage. In many cases the interface need not be as extensive as shown here and results in saving board space. Two cases where considerable saving is achieved are:

Case 1: System Using 8288

If the system uses an 8288 bus controller for 80186, pre-processing of $\overline{WR}$ input is not necessary and the $\overline{IOWC}$ output of 8288 can be fed directly to pin 5 of U2 (74LS08). This is because $\overline{IOWC}$ signal meets the timing requirements of the SCC. Also note, that the interface circuit is then totally independent of the 80186 clock.

Case 2: System Using Non-Vectored Interrupt Mode for SCC

Such a system will not need the component U1 (74LS74) nor the AND gate U2 (pins 11, 12, 13). Pin 3 of U2 can be fed directly to the $\overline{RD}$ input of SCC.

NOTE:

Worst Case Timings. The four TTL interface solution is based on typical values. It is *not* based on worst-case timings. The exact circuit design may vary depending on the 80186 and 82530 clock frequencies and the worst case timings of the TTL packages.

CONCLUSION

This four TTL package interface solution is low cost and compact (1.2 sq. inch). It should satisfy 82530 interfacing for almost all applications. In fact, as already mentioned, many applications may require only 2–3 TTL packages for interfacing the 82530 to 80186 or to other INTEL processors.

intel®

APPLICATION NOTE

AP-401

December 1986

Designing With the 82510 Asynchronous Serial Controller

FAISAL IMDAD-HAQUE
APPLICATIONS ENGINEER

Order Number: 231928-001

1.0 INTRODUCTION

The emergence of asynchronous communications as the most widely used protocol (it commands the largest installed base of nodes, exceeding HDLC/SDLC, the second most popular protocol, by a factor of 10 to 1) for point to point serial links has led to the need for an asynchronous communications component with high integration to reduce component count and decrease the cost of a serial port. The trend towards higher data rates and multiple job, multiple user systems has underscored the need for an intelligent serial controller to improve system throughput and decrease the CPU load normally associated with asynchronous serial communications.

The 82510 CMOS Asynchronous Serial Controller is designed to improve asynchronous communications throughput and reduce system cost by integrating functions and simplifying the system interface. Two independent FIFOs, and Control Character Recognition (CCR), provide data buffering and increase software efficiency. Two Baud Rate Generators/Timers, an On-Chip Crystal Oscillator and seven Programmable I/O pins provide a high degree of integration and reduce system component count. This application note will demonstrate the use of these features in an Asynchronous Communications Environment.

1.1 Goal

The goal of this application note is to demonstrate the use of the major 82510 features in an asynchronous communications environment and to depict basic hardware and software design techniques for the 82510. It will discuss interfaces using both polling and interrupt techniques, as well as the impact of FIFOs using either scheme. An application example covering the application of Error Free File Transfer is also provided.

1.2 Scope

The application note describes the operation of the 82510 ASC in a normal (non 8051 9-bit) asynchronous communications mode. The majority of the discussion is focused towards the systems aspects of the Controller. The use of the 82510 in a multidrop or 8051 9-bit asynchronous environment is not covered. This application note assumes that the reader is familiar with the 82510 in terms of pin description, register architecture and interrupt structure. It is also assumed that the reader is familiar with the information provided in the 82510 Data Sheet.

The initial sections of the application note provide an overview of the 82510 and its major functional blocks. This is followed by a discussion of the hardware design and system interface considerations in sections three and four. The fifth section provides some software techniques for transmitting and receiving data as well as the use of timers. Section seven briefly discusses the file transfer application based on the XMODEM protocol and includes the software listings.

2.0 82510 DESCRIPTION

2.1 Overview

The 82510 can be divided into seven functional blocks (See Figure 1): Bus Interface Unit (BIU), Timing Unit, Modem Interface Module, Tx FIFO, Rx FIFO, Tx Machine and Rx Machine. All blocks, except BIU can generate a block interrupt request to the 82510 interrupt logic. In the case of the Rx Machine, Timing Unit and Modem Interface Module, multiple sources (errors and status events) within the block cause the block interrupt request to become active. All of the blocks have registers associated with them. The registers, allow configuration, provide status information about events/errors, and may also be used to send commands to each block.

2.2 Bus Interface Unit (BIU)

The Bus Interface Unit (BIU) interfaces the 82510 functional blocks to the system or CPU bus. It provides read and write access to the 82510 registers and controls the generation of interrupts to the external world. The interrupt logic resolves contention between block interrupt requests, on a priority basis. The BIU also has the Hardware Reset circuitry, which is driven by the RESET pin. The reset signal clears all internal Flip Flops, and Registers and puts them in a predefined state. All activities on the Bus interface, including register accesses by the CPU, are synchronized to an internal (82510) system clock, supplied via the CLK pin.

2.3 Receive Machine (RxM)

The Rx Machine (RxM) converts the serial data to parallel and writes it to the Rx FIFO, along with the appropriate flags (available in the *Receive Flags Register*). The Rx Machine can be configured for control character recognition, data sampling and DPLL operation. The software can check for noise, control character, break, address or parity and framing errors by reading the status or character flags. Optionally, the Receive Status bits (in RST), when enabled, can generate interrupt requests. The Rx Machine block Interrupt request is reflected in the *General Status Register* and is set when an enabled interrupt request within the Rx Ma-

chine (i.e. RST bits) becomes active. The Rx Machine has eight registers associated with it:

Receive Data (RXD)—Receive Data Character

Receive Flags (RXF)—Receive Character Flags

Receive Status (RST)—Receive Events and Receive Errors

Receive Interrupt Enable (RIE)—Enables Interrupts on corresponding bits in RST

Receive Mode (RMD)—Receive Machine Configuration

Receive Command (RCM)—Receiver Command Register

Line Control (LCR)—16450 Register, Character Attribute Configuration

Line Status (LSR)—16450 Status Register, Tx and Rx status

2.4 Transmit Machine (TxM)

The Tx Machine reads characters from the Tx FIFO and transmits them serially over the TXD line. The Tx Machine can also transmit additional character attributes (9th bit of Data, Address Marker, Software Parity) available from the Transmit Flags, if configured in the appropriate mode. The Tx Machine Idle interrupt request is reflected in the GSR and LSR registers to indicate that the Transmitter is either Empty or Disabled. The Tx Machine has six registers associated with it:

Line Control (LCR)—16450 Register, Character Attribute Configuration

Line Status (LSR)—16450 Status Register, Tx and Rx status

Transmit Mode (TMD)—Tx Machine Configuration

Transmit Command—(TCM)—Transmit Command Register

Transmit Flags (TXF)—Transmit Character Flags

Transmit Data (TXD)—Transmit Data Character

2.5 Modem Interface Module

The Modem Interface module is responsible for the modem interface and general purpose I/O pins. It will generate Interrupts (if enabled) upon transitions in the modem input pins ($\overline{DCD}$, $\overline{CTS}$, $\overline{RI}$, and $\overline{DSR}$). The modem output pins can be controlled by the CPU, also the RTS pin can be used to provide flow control, in the automatic transmission mode. It is the source of the Modem Interrupt bit in GSR. This bit is set whenever there is a state change in the $\overline{DCD}$, $\overline{RI}$, $\overline{DSR}$ or $\overline{CTS}$ inputs (reflected in *Modem Status Register*) and the corresponding enable bits are set. The function and direction of the multifunction pins can be reprogrammed and is available as a configuration option. Multifunction pins, when configured as outputs, can be controlled by the CPU through the *Modem Control Register*. The Modem module has four registers associated with it:

Modem Status
Register (MSR)—State transitions on modem input pins, and State of the modem input pins

Modem Control (MCR)—Control state of Modem Output pins

Modem Interrupt
Enable (MIE)—Enable Interrupt on State transitions in modem input pins

I/O Pin Mode (PMD)—Functions and Directions of Multifunction pins

2.6 Timing Unit

The Timing Unit is responsible for the generation of the System Clock, using either its Crystal Oscillator or an externally generated clock, and generation of the Tx and Rx clocks from either the On-Chip Baud Rate Generators or the SCLK pin. It is also responsible for generating Timer Expired interrupts when the BRGs/Timers are configured for use as Timers. There are ten registers associated with the Timing Unit, four of these are used in the Timer mode only.

Timer Status (TMST)—Timer A and/or Timer B expired

Timer Interrupt
Enable (TMIE)—Enables Interrupts upon Expiration of Timers A or B in TMST

Timer Control (TMCR)—Start and Disable Timers

Clock Configure (CLCF)—Select source and mode for Tx and Rx clocks

BRG B Configuration (BBCF)—Mode and Clock source of BRG B

BRG B LSB of Divisor (BBL)—Least Significant Byte of BRG B Divisor/Count

BRG A MSB of Divisor (BBH)—Most Significant Byte of BRG B Divisor/Count

BRG A Configuration (BACF)—Mode and Clock source of BRG A

BRG A LSB of Divisor (BAL)—Least Significant Byte of BRG A Divisor/Count

BRG A MSB of Divisor (BAH)—Most Significant Byte of BRG A Divisor/Count

2.7 FIFOs, Rx and Tx

The Dual FIFOs (transmit and receive), serve as buffers for the 82510. They buffer the transmitter and Receiver from the CPU. Each of the FIFOs has a programmable threshold. The threshold is the FIFO level which will generate an interrupt. The threshold is used to optimize the CPU throughput and provide increased interrupt to service latency for higher baud rates. It can be configured through the FIFO Mode Register. Each FIFO character has flags associated with it (TxF and RxF). As each character is read from the Rx FIFO its flags are put into the RxF register. Before a write to TXD (if character configuration requires) the character flags are written to the TXF register. The two FIFOs are totally independent of each other and each FIFO can generate an interrupt request which indicates that the configured threshold has been met.

3.0 HARDWARE DESIGN

3.1 System Interface

The 82510 has a standard I/O peripheral interface, it has a demultiplexed Bus, which consists of a bidirectional eight bit Data Bus, and three Address lines. Interrupt, Read, Write, Chip Select and Reset pins complete the system interface. The three address lines along with the *Bank register* are used to select a particular register.

3.1.1 REGISTER ACCESS

The 82510 registers are logically divided into four banks. Only one bank can be accessed at any one time. Each register bank occupies eight I/O addresses. To select a register, the correct Bank must first be selected by writing to the GIR/Bank register (the *GIR/Bank register* I/O address is two ($A_0 = 0, A_1 = 1, A_2 = 0$). Then one of the eight I/O space addresses is selected by outputting a value (between zero and seven) to the 82510 address pins A_0–A_2.

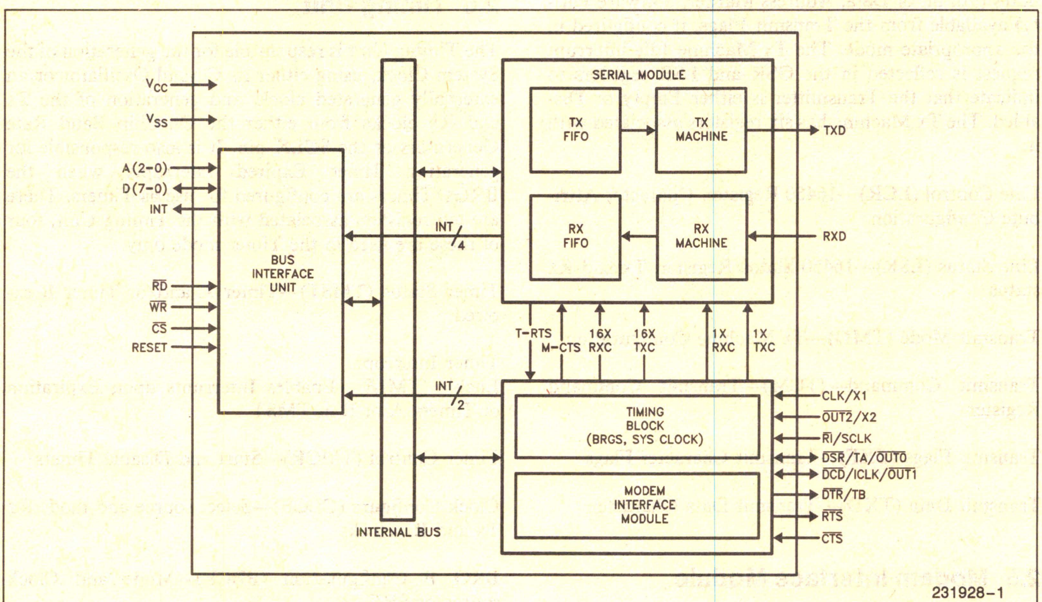


Figure 1. 82510 Block Diagram

BANK ZERO 8250—COMPATIBLE BANK										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
TxD	Tx Data bit 7	Tx Data bit 6	Tx Data bit 5	Tx Data bit 4	Tx Data bit 3	Tx Data bit 2	Tx Data bit 1	Tx Data bit 0	0	—
RxD	Rx Data bit 7	Rx Data bit 6	Rx Data bit 5	Rx Data bit 4	Rx Data bit 3	Rx Data bit 2	Rx Data bit 1	Rx Data bit 0	0	—
BAL	BRGA LSB Divide Count (DLAB = 1)								0	02H
BAH	BRGA MSB Divide Count (DLAB = 1)								1	00H
GER	0	0	Timer Interrupt Enable	Tx Machine Interrupt Enable	Modem Interrupt Enable	Rx Machine Interrupt Enable	Tx FIFO Interrupt Enable	Rx FIFO Interrupt Enable	1	00H
GIR/BANK	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
LCR	DLAB Divisor Latch Access bit	Set Break	Parity Mode bit 2	Parity Mode bit 1	Parity Mode bit 0	Stop bit Length bit 0	Character Length bit 1	Character Length bit 0	3	00H
MCR	0	0	OUT 0 Complement	Loopback Control bit	OUT 2 Complement	OUT 1 Complement	RTS Complement	DTR Complement	4	00H
LSR	0	TxM Idle	Tx FIFO Interrupt	Break Detected	Framing Error	Parity Error	Overrun Error	Rx FIFO Int Reg	5	60H
MSR	DCD Input Inverted	RI Input Inverted	DSR Input Inverted	CTS Input Inverted	State Change in DCD	State (H → L) Change in RI	State Change in DSR	State Change in CTS	6	00H
ACR0	Address or Control Character Zero								7	00H

BANK ONE—GENERAL WORK BANK										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
TxD	Tx Data bit 7	Tx Data bit 6	Tx Data bit 5	Tx Data bit 4	Tx Data bit 3	Tx Data bit 2	Tx Data bit 1	Tx Data bit 0	0	—
RxD	Rx Data bit 7	Rx Data bit 6	Rx Data bit 5	Rx Data bit 4	Rx Data bit 3	Rx Data bit 2	Rx Data bit 1	Rx Data bit 0	0	—
RxF	—	Rx Char OK	Rx Char Noisy	Rx Char Parity Error	Address or Control Character	Break Flag	Rx Char Framing Error	Ninth Data bit of Rx Char	1	—
TxF	Address Marker bit	Software Parity bit	Ninth bit of Data Char	0	0	0	0	0	1	—
GIR/BANK	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
TMST	—	—	Gate B State	Gate A State	—	—	Timer B Expired	Timer A Expired	3	30H
TMCR	0	0	Trigger Gate B	Trigger Gate A	0	0	Start Timer B	Start Timer A	3	—
MCR	0	0	OUT 0 Complement	Loopback Control bit	OUT 2 Complement	OUT 1 Complement	RTS Complement	DTR Complement	4	00H

Figure 2. 82510 Register Map

BANK ONE—GENERAL WORK BANK (Continued)										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
FLR	—	Rx FIFO Level			—	Tx FIFO Level			4	00H
RST	Address/ Control Character Received	Address/ Control Character Match	Break Terminated	Break Detected	Framing Error	Parity Error	Overrun Error	Rx FIFO Interrupt Requested	5	00H
RCM	Rx Enable	Rx Disable	Flush RxM	Flush Rx FIFO	Lock Rx FIFO	Open Rx FIFO	0	0	5	—
MSR	DCD Complement	RI Input Inverted	DSR Input Inverted	CTS Input Inverted	State Change in DCD	State Change in RI	State Change in DSR	State Change in CTS	6	00H
TCM	0	0	0	0	Flush Tx Machine	Flush Tx FIFO	Tx Enable	Tx Disable	6	—
GSR	—	—	Timer Interrupt	TxM Interrupt	Modem Interrupt	RxM Interrupt	Tx FIFO Interrupt	Rx FIFO Interrupt	7	12H
ICM	0	0	0	Software Reset	Manual Int Acknowledge Command	Status Clear	Power Down Mode	0	7	—

BANK TWO—GENERAL CONFIGURATION										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
FMD	0	0	Rx FIFO Threshold		0	0	Tx FIFO Threshold		1	00H
GIR/BANK	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
TMD	Error Echo Disable	Control Character Echo Disable	9-bit Character Length	Transmit Mode		Software Parity Mode	Stop Bit Length		3	00H
IMD	0	0	0	0	Interrupt Acknowledge Mode	Rx FIFO Depth	ulan Mode Select	Loopback or Echo Mode of Operation	4	0CH
ACR1	Address or Control Character 1								5	00H
RIE	Address/ Control Character Recognition Interrupt Enable	Address/ Control Character Match Interrupt Enable	Break Terminate Interrupt Enable	Break Detect Interrupt Enable	Framing Error Interrupt Enable	Parity Error Interrupt Enable	Overrun Error Interrupt Enable	0	6	1EH
RMD	Address/Control Character Mode		Disable DPLL	Sampling Window Mode	Start bit Sampling Mode	0	0	0	7	00H

BANK THREE—MODEM CONFIGURATION										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
CLCF	Rx Clock Mode	Rx Clock Source	Tx Clock Mode	Tx Clock Source	0	0	0	0	0	00H
BACF	0	BRGA Clock Source	0	0	0	BRGA Mode	0	0	1	04H
BBL	BRGB LSB Divide Count (DLAB = 1)								0	05H
BBH	BRGB MSB Divide Count (DLAB = 1)								1	00H

Figure 2. 82510 Register Map (Continued)

BANK THREE—MODEM CONFIGURATION (Continued)										
Register	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Address**	**Default**
GIR/BANK	0	BANK Pointer bit 1	BANK Pointer bit 0	0	Active Block Int bit 2	Active Block Int bit 1	Active Block Int bit 0	Interrupt Pending	2	01H
BBCF	BRGB Clock Source		0	0	0	BRGB Mode	0	0	3	84
PMD	DCD/ICLK/ OUT 1 Direction	DCD/ICLK/ OUT 1 Function	DSR/TA/ OUT 0 Direction	DSR/TA/ OUT 0 Function	RI/SCLK Function	DTR/TB Function	0	0	4	FCH
MIE	0	0	0	0	DCD State Change Int Enable	RI State Change Int Enable	DSR State Change Int Enable	CTS State Change Int Enable	5	0FH
TMIE	0	0	0	0	0	0	Timer B Interrupt Enable	Timer A Interrupt Enable	6	00H

Figure 2. 82510 Register Map (Continued)

3.1.2 READ AND WRITE CYCLES

Like most other I/O based peripherals the Read and Write pins are used to access data in the 82510. Each read or write cycle has specified setup and hold times in order for the data to be transferred correctly to/from the 82510. The critical timings for the read cycle are:

1. Address Valid to Read Active (Tavrl)
2. Command Access Time to Data Valid (Trldv)
3. Command Active Width (Trlrh)

The less critical parameters are:

4. Address Hold to Read Inactive (Trhax)
5. Data Out Float Delay after Read Inactive (Trhdz)

The critical timings for the write cycle are:

1. Address Valid to Write Low (Tavwl)
2. Write Active Time (Twlwh)
3. Data Valid to Write Inactive (Tdvwh)

The less critical parameters are:

4. Address and Chip Select Hold Time After Write Inactive (Twhax)
5. Data Hold Time After Write Inactive (Twhdx)

These timings determine the number of wait states required for the 82510 and the CPU interface. The interfaces for some popular microprocessors are discussed in the following sections.

3.1.3 80186 INTERFACE

The exact interface is shown in Figure 3. The schematic shows the 80186 interface to the 82510 on a local bus. Although the Data Bus is buffered, it is possible to directly connect the 82510 to the 80186 data bus; because the Data Float Delay after read inactive is 40 ns for the 82510, which is well under the 85 ns requirement of the 80186. The timing equations for the interface are given below.

Read Cycle:

Address to Read Low $= \text{Tclcl} - \text{Tclav}_{max} + \text{Tclrl}_{min} - \text{Latch Delay}_{max}$

Read Access Time $= 2\text{Tclcl} - \text{Tclrl}_{max} - \text{Tdvcl} - \text{Transceiver Delay}_{max}$

Read Active Time $= \text{Trlrh}$
$= 2\text{Tclcl} - 46$

Write Cycle:

Address Valid to Write Active $= \text{Tclcl} + \text{Tcvctv}_{min} - \text{Tclav}_{max} - \text{Latch Prop. Delay}_{max}$

Write Active Time $= \text{Twlwh}$
$= 2\text{Tclcl} - 40$

Data Valid to Write Inactive $= 2\,\text{Tclcl} - \text{Tcldv}_{max} - \text{Transceiver Delay}_{max} + \text{Tcvctx}_{min}$

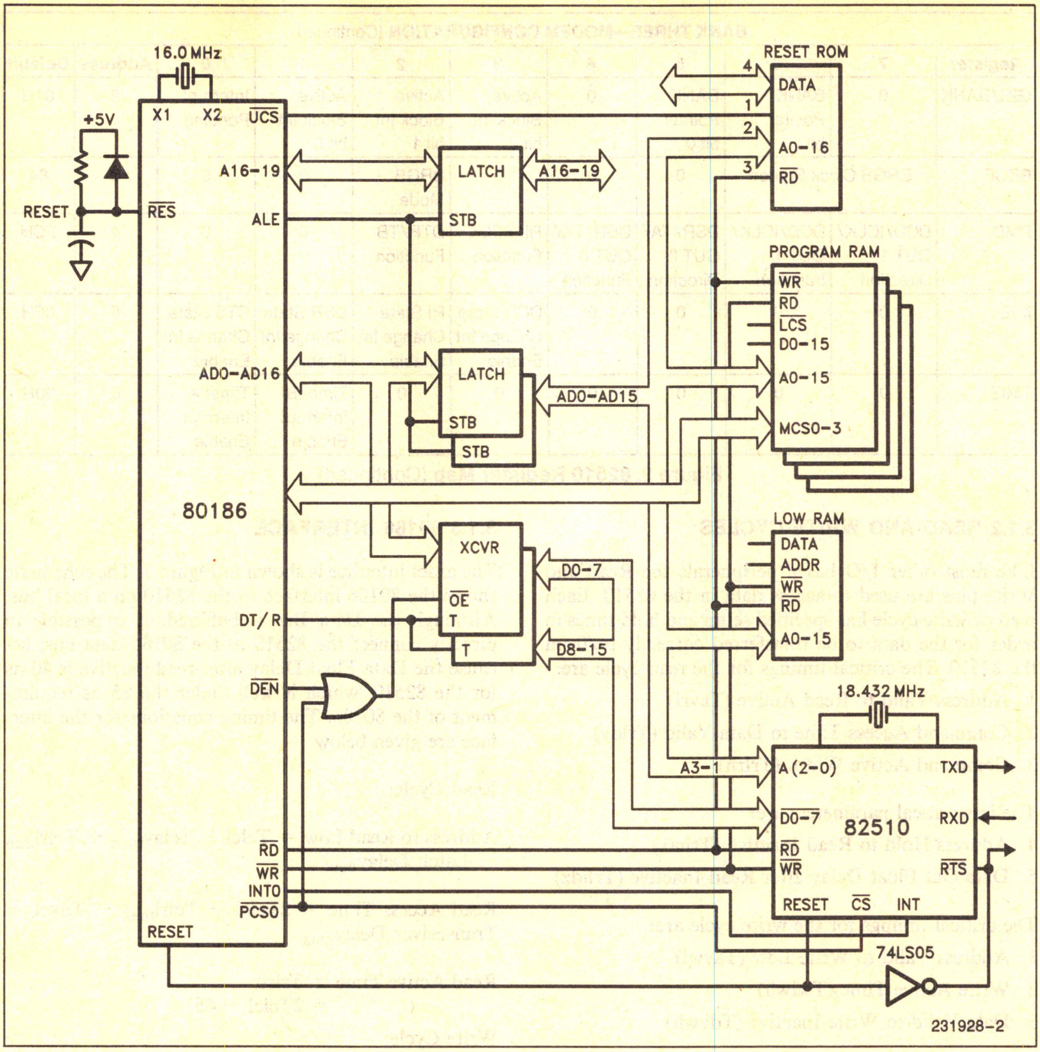


Figure 3. 82510 Interface to 80186

The user can transfer data to the 82510, using the DMA capabilities of the 80186, by using the $\overline{RTS}$ pin, in automatic modem control mode, as a DMA request line. The $\overline{RTS}$ pin, in automatic mode, will go inactive as soon as the Tx FIFO and the Tx shift register are empty. It will become active once a data character is written to the *TXD register.* In most 80186 DMA transfers the user has to make sure that the DMA request line goes inactive at least two clock cycles from the end of the DMA deposit cycle. In this case, the extra DMA cycle is not a problem, because the Tx FIFO will buffer the data to prevent an overrun (Since the Tx FIFO can buffer up to four characters, the $\overline{RTS}$ pin only needs to go inactive two clocks before the end of the deposit phase of the fourth DMA). Typically $\overline{RTS}$ will go inactive five (82510) system clocks after the rising edge of write.

3.1.4 80286 INTERFACE

The 80286 interface is shown in Figure 4. The 82510 is on the local bus, and is using the control signals from the 82288 Bus Controller. The Data Enable (OE) is qualified by the 82510 Chip Select, to avoid Data Bus contention between the 82510 and the CPU. The timing equations for the Read and Write Cycles are given below.

Read Cycle:

Address Valid to Read Active = T1 (CLK period) + $T29_{min}$ (CLK to cmd active) − $T16_{max}$ (ALE active delay) − Latch Prop. $Delay_{max}$

Read Access to Data Valid = 2T1 (CLK period) − $T29_{max}$ (CLK to cmd active) − T8 (Read Data Setup Time) − Transceiver $Delay_{max}$

Read Active Time = 2T1 (CLK period) − $T29_{max}$ (CLK to cmd active) + $T30_{min}$ (CLK to cmd inactive)

Write Cycle:

Address to Write Low = T1 (CLK period) + $T29_{min}$ (CLK to cmd active) − $T16_{max}$ (ALE active delay) − Latch $Delay_{max}$

Write Active Time = 2 T1 (CLK period) − $T29_{max}$ (CLK to cmd active) + $T30_{min}$ (CLK to cmd inactive)

Data to Write High = 3 T1 − $T14_{min}$ (Write Data Valid Delay) + $T30_{min}$ (CLK to cmd inactive) − Xcvr. $Delay_{max}$

Using an 8 MHz 80286 with the 82510 at 18.432 MHz (divide by two—9.216 MHz) requires two wait states. The critical timings are the read cycle timings—Read Access Time and Read Active Width. Inserting two wait states means that the access times for the relevant parameters will be increased by 250 ns.

NOTE:
The address decoding scheme of the 80286 interface is different from the IBM PC/PC AT I/O addresses for the serial ports, therefore the interface shown in Figure 4 cannot be used in PC/PC AT oriented designs.

3.1.5 80386 INTERFACE

The 80386 interface to the 82510 is given in Figure 5. The example uses the Basic I/O interface given in the 80386 Hardware Reference Manual section 8.3. The only differences are in the specific address lines used for chip select generation, and the additional wait states in the wait state generation logic. The address lines A3, A4 and A5 are used to select one of the eight register address spaces in the 82510, therefore, A6 and A7, rather than A4 and A5, are used in the I/O decoder. This causes a granularity of four in the 82510's I/O address space, i.e., the addresses of two consecutive registers in the 82510 differ by four.

The 82510 requires one additional wait state (as currently specified), the design assumes that the PAL equations are modified for that purpose. The user may also externally generate the wait states and connect to the "other ready logic" input ORed with the RDY pin of PAL 2. The two read timings Read Active width and Read Access time to Data Valid each require one additional wait state in order to meet the 82510 timing requirements. The timings are given below. (82510 times are at 9.216 MHz)

Read Cycle:

Read Access to Data Valid	= 253.25 ns
82510 Trldv	= 308
additional time reqd.	= 308-253.25
	= 54.75 ns
Read Active Width	= 269.25
82510 Trlrh	= 308
additional time reqd.	= 308-269.25
	= 38.75 ns
Address Valid to Read Active	= 132.75 ns
82510 T_{AVRL}	= 7 ns

Since each additional wait state adds 62.5 ns at 16 MHz, the 82510 requires one additional wait state.

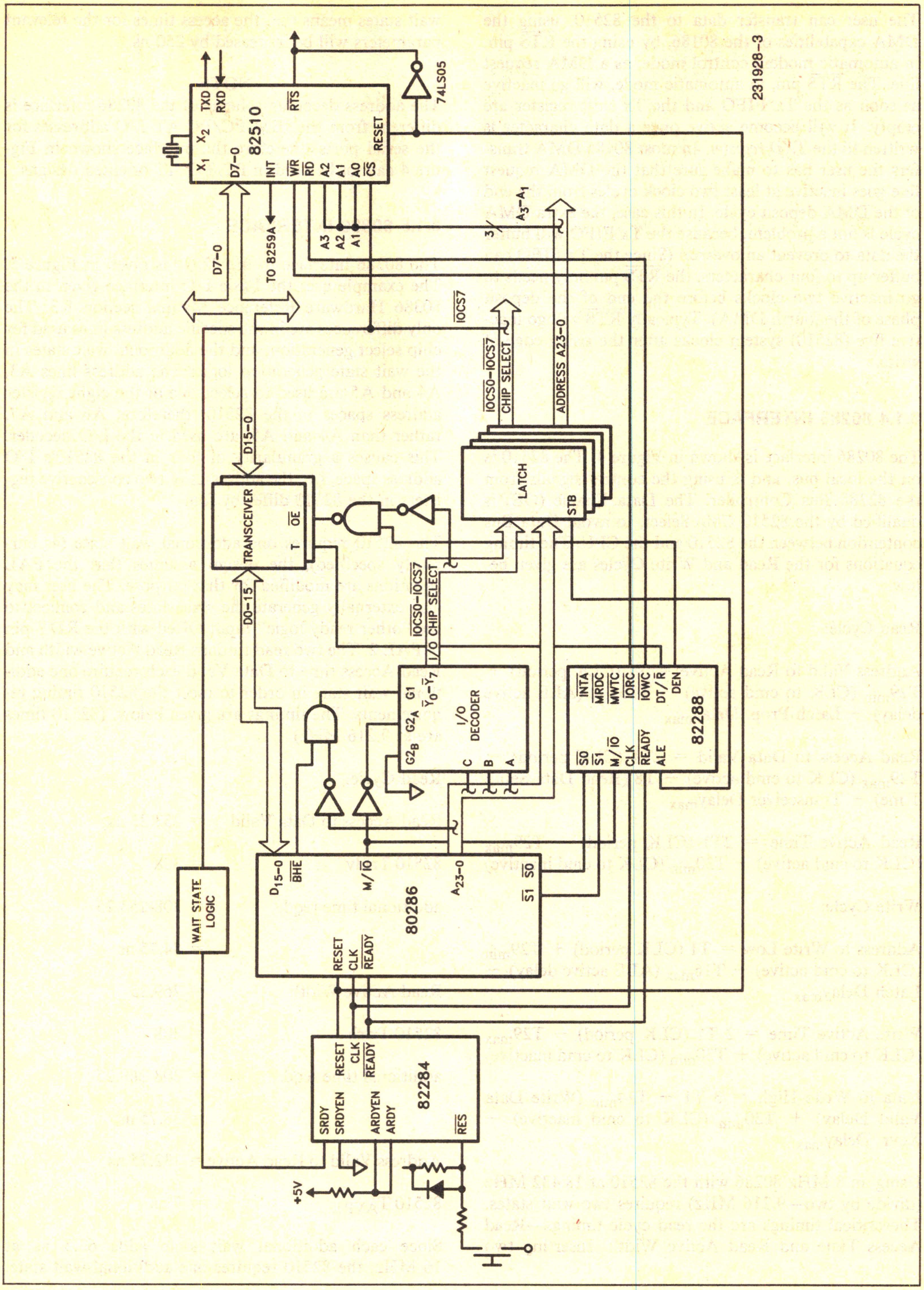


Figure 4. 80286 Interface to the 82510

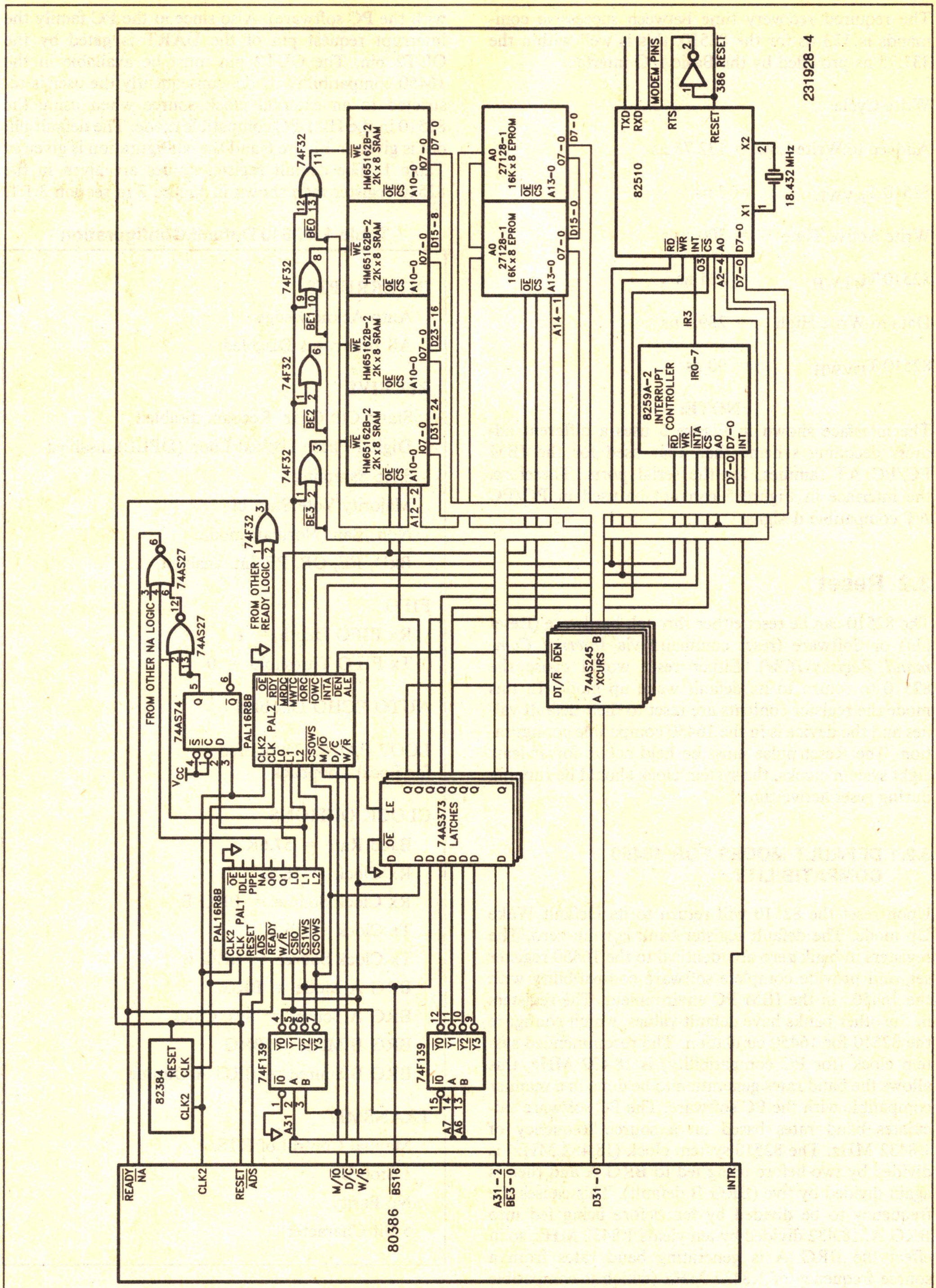


Figure 5. 80386 Interface to the 82510

The required recovery time between successive commands is 123 ns for the 82510, this is well within the 331.75 ns provided by the Basic I/O interface.

Write Cycle:

Addven to Write Low	= 132.75 ns
82510 T_{AVWL}	= 7 ns
Write Active Time	= 300.5 ns
82510 T_{WLWH}	= 231 ns
Data to Write High	= 289.5 ns
82510 T_{DVWH}	= 90 ns

NOTE:
The interface shown in Figure 5 uses a different address decoding scheme than that used for the IBM PC/PC AT families, for the serial ports. Therefore, the interface in Figure 5 can not be used in PC/PC AT compatible designs.

3.2 Reset

The 82510 can be reset either through hardware (Reset pin) or Software (reset command via *Internal Command Register-ICM*). Either reset would cause the 82510 to return to its default wake up mode. In this mode the register contents are reset to their default values and the device is in the 16450 compatible configuration. The Reset pulse must be held active for at least eight system clocks, the system clock should be running during reset active time.

3.2.1 DEFAULT MODES FOR 16450 COMPATIBILITY

Upon reset the 82510 will return to its Default Wake Up mode. The default register bank is bank zero. The registers in bank zero are identical to the 16450 register set, and provide complete software compatibility with the 16450* in the IBM PC environment. The registers in the other banks have default values, which configure the 82510 for 16450 emulation. The recommended system clock (for PC compatibility) is 18.432 MHz, this allows the baud rates generation to be done in a manner compatible with the PC software. The PC software calculates baud rates based on a source frequency of 1.8432 MHz. The 82510 system clock (18.432 MHz) is divided by two before being fed to BRG A and then is again divided by five (BRG B default). This causes the frequency to be divided by ten before being fed into BRG A. 18.432 divided by ten yields 1.8432 MHz, so in effect the BRG A is generating baud rates from a source frequency of 1.8432 MHz (which is compatible with the PC software). Also since in the PC family the interrupt request pin of the UART is gated by the OUT2 pin, The OUT2 pin must be available in the 16450 compatibility mode, consequently the user is restricted to an external clock source when using the 82510 in the IBM PC compatible mode. The default pin out is given in Figure 6 and the configuration is given in Table 1. The default register values are given in the 82510 register map shown in Figure 2 in section 3.1.1.

*16450 is the PC AT version of the INS 8250A.

Table 1. 82510 Default Configuration

INTERRUPTS
Auto Acknowledge
All Interrupts Disabled
RECEIVE
Stand Ctl. Char. Recogn. disabled
Digital Phase Locked Loop (DPLL) disabled
3/16 Sampling
Majority Vote Start bit
Non μlan (Normal) mode
BkD, FE, OE, PE Int. enabled
FIFO
Rx FIFO Depth = 1
Tx FIFO Threshold = 0
AUTO ECHO Disabled
LOOP BACK Configured for Local Loopback
CLOCK OPTIONS
Baud Rate = 57.6K
Rx Clock = 16 x
Rx Clock Source = BRG B
Tx Clock = 16 x
Tx Clock Source = BRG B
BRG A Mode = BRG
BRG A Source = Sys. Clock
BRG B Mode = BRG
BRG B Source = BRG A Output
TRANSMIT
Manual Control of RTS
1 Stop Bit
No Parity
5 Bit Character

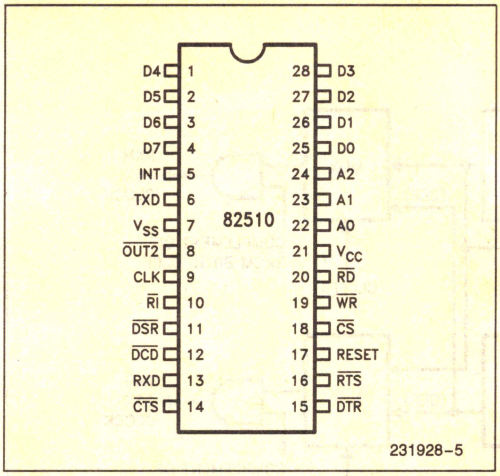
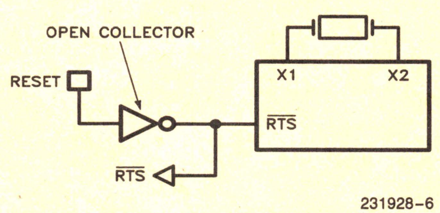


Figure 6. Default Pin Out Configuration of the 82510

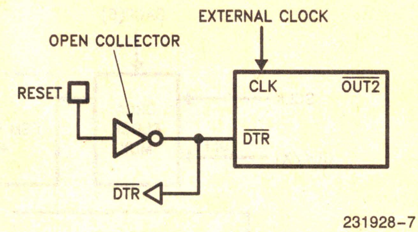
3.3 System Clock Options

The term "System Clock" refers to the clock which provides timings for most of the 82510 circuitry. The 82510 has two modes of system clock usage. It can generate its system clock from its On-Chip Crystal Oscillator and an external crystal, or it can use an externally generated clock, input to the device through the CLK pin. The selection of the system clock option is done during reset. The default system clock source is an externally generated clock, which can be reconfigured by a strapping option on the $\overline{RTS}$ pin. During Reset, the $\overline{RTS}$ pin is an input; it is internally pulled high, if it is externally driven low, then the 82510 expects to use the Crystal Oscillator for system clock generation, otherwise it is set up for using an external clock source. This can be done by using an open collector inverter to $\overline{RTS}$, the input of the inverter is the Reset signal. The 82510 has a pull up resistor in the $\overline{RTS}$ circuitry so no external pull up is needed. In the crystal oscillator mode the CLK/X1 pin is automatically configured to X1, and the $\overline{OUT2}$/X2 pin is configured to X2. In the External Clock mode, the CLK/X1 is configured to CLK and the $\overline{OUT2}$/X2 is configured to $\overline{OUT2}$.



1 ms is needed for Oscillator startup

Figure 7. Crystal Oscillator Strapping Option



NOTE:
Crystal Oscillator is always divided by two.

Figure 8. Disable Divide by Two

If the Crystal Oscillator is being used to supply the system clock, then the clock frequency is always divided by two before being fed into the rest of the 82510 circuitry. If, however an external clock source is being used to supply the system clock, then the user has two options:

1. Use the System Clock after **division by two,** e.g. if a 8 MHz clock is being fed into the CLK pin, then the actual frequency of the 82510 system clock will be 4 MHz (default).
2. **Disable Division by two** and use the direct undivided clock, e.g. if an 8 MHz clock is being fed into the CLK pin, then the actual frequency of the 82510 system clock is also 8 MHz.

The divide by two option is the default mode of operation in the External Clock mode of the 82510. A strapping option can be used to disable the Divide By Two operation (For Crystal Oscillator Mode Divide By Two must always be active). During Reset, the $\overline{DTR}$ pin is an input; it is internally pulled high, if it is externally driven low then the Divide By Two operation is disabled. The strapping option is identical to the one used on $\overline{RTS}$ for selection of the System Clock source.

The 82510 system clock must be chosen with care since it influences the wait state performance, Baud Rate Generation (if being used as source frequency for the BRGs), the power consumption, and the Timer counting period. The power consumption of the 82510 is dependent upon the system clock frequency. If using the system clock as a source for the Baud Rate Generator(s), then the system clock frequency must be a baud rate multiple in order to minimize frequency deviation. For standard baud rates a multiple of 1.8432 MHz can be used, in fact the 18.432 MHz maximum frequency was chosen with this particular criteria in mind.

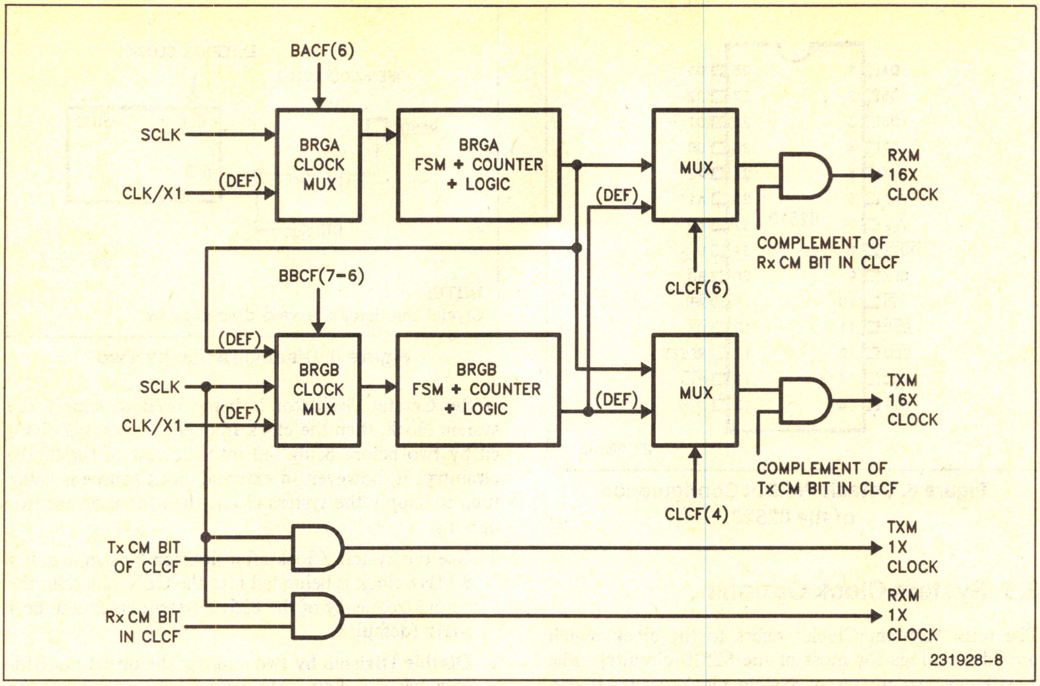


Figure 9. Timing Flow of the 82510

3.3.1 POWER DOWN MODE

The 82510 has a "power down" mode to reduce power consumption when the device is not in use. The 82510 powers down when the power down command is issued via the *Internal Command Register* (ICM). There are two modes of power down, Power Down Sleep and Power Down Idle.

3.3.1.1 Sleep Mode

This is the mode when even the system clock of the 82510 is shut down. The system clock source of the 82510 can either be the Crystal Oscillator or an external clock source. If the Crystal Oscillator is being used and the power down command is issued, then the 82510 will automatically enter the Sleep mode. If an external clock is being used, then the user must disable the external clock in addition to issuing the Power Down command, to enter the Sleep mode. The benefit of this mode is the increased savings in power consumption (typical power consumption in the Sleep mode is in the range of hundreds of microAmps. However, upon wake up, if using a crystal oscillator, the user must reprogram the device. The data is preserved if the external clock is disabled after the power down command, and enabled prior to exiting the power down mode. To exit this mode the user can either issue a Hardware reset, or read the *FIFO Level Register (FLR)* and then issue a software reset (if using a Crystal Oscillator). In either case the contents of the 82510 registers are not preserved and the device must be reprogrammed prior to operation.

NOTE:
If the Crystal Oscillator is being used then the user must allow about 1 ms for the oscillator to wake up before issuing the software reset.

3.3.1.2 Idle Mode

The 82510 is said to be in the Idle mode when the Power Down command is issued and the system clock is still running (i.e. the system clock is generated externally and not disabled by the user). In this mode the contents of all registers and memory cells are preserved, however, the power consumption in this mode is greater than in the Sleep mode. Reading FLR will take the 82510 out of this mode.

NOTE:
The data read from FLR when exiting Power Down is incorrect and must be ignored.

4.0 INTERRUPT BEHAVIOR

4.1 FIFO Usage

The 82510 has two independent four bytes transmit and receive FIFOs. Each FIFO can generate an interrupt request, when the FIFO level meets the Threshold requirements. The FIFOs can have a considerable impact on the performance of an asynchronous communications system. For systems using high baud rates they can provide increased interrupt-to-service latency reducing the chances of an overrun occurring. In systems constrained for CPU time, the FIFOs can increase the CPU Bandwidth by reducing the number of interrupt requests generated during asynchronous communications. It can reduce the interrupt load on the CPU by up to 75%. By choosing the FIFO thresholds which reflect the system bandwidth or service latency requirements, the user can achieve data rates and system throughput, unattainable with traditional UARTs.

Table 2. The Power Down Modes

Mode	Clock Source	Exit Procedure	Power Consumption	Data Preservation
Sleep	Crystal Oscill. Automatically Disabled	H/W Reset or Read FLR and Issue S/W Reset	100–900 μA	Not Preserved Must be Reprogrammed
	External Clock Must be Disabled by User	Enable External Clock, Read FLR and Issue S/W Reset H/W Reset	100–900 μA	Not Preserved Must be Reprogrammed
Idle	External Clock Running	H/W Reset Read FLR	1–3 mA	All Data Preserved Does Not Need to be Reprogrammed

4.1.1 INTERRUPT-TO-SERVICE LATENCY

The interrupt-to-service latency is the time delay from the generation of an interrupt request, to when the interrupt source in the 82510 is actually serviced. Its primary application is in the reception of data. In traditional UARTs the CPU must read the current character in the Receive Buffer before it is overrun by the next incoming character. The Rx FIFO in the 82510 can buffer up to four characters, allowing an interrupt-to-service latency of up to four character transmission times. The character transmission time is the time period required to transmit one full character at the given Baud Rate. It is dependent upon the baud rate and is given by equation (1):

(1) Character Transmission Time =

$$\frac{\text{Num. of Bits per Character Frame}}{\text{Baud Rate}}$$

The Transmit and Receive FIFO thresholds should be selected with consideration to two factors the Baud rate, and the (CPU Bandwidth allocated for Asynchronous Channels is dependent upon the number of channels supported since it does not include the overhead of supporting other peripherals) number of Asynchronous Serial ports being supported by the CPU. In order to avoid overrun, the interrupt-to-service delay must be less than the time it takes to fill the 82510 Rx FIFO. The relationship is given by equation (2):

(2) Int__to__service-latency < FIFO Size × Character Transmission Time

Example

Calculate the maximum baud rate that can be supported by a 6 MHz PC AT to support four Full Duplex Asynchronous channels using

a) The 82510 with four byte FIFO.

b) The 82510 with one byte FIFO.

Assumptions:

- CPU dedicated to Asynchronous communications.
- UART Interrupts limited to Transmission and Reception only.
- Interrupt Routines are optimized for fast throughput.
- 10 bits per character frame.

Going back to equation (2):

Int.__to__service latency < Buffer size x 10/baud rate

Int__to__service latency = # of Channels × (# of int. sources per channel) × Time required to service interrupt

Int__to__service latency = 4 × 2 × Time required to service interrupt

The Time required to service interrupt has been calculated to be 100 μs for a slightly optimized service routine. RMX86 interrupt service time is given as 250 μs and for other operating systems it should be slightly higher.

Int__to__service latency	= 4x2x100 s
	= 800 μs
82510 max Baud Rate	= 4 × 10/800 μs
(four byte FIFO)	= 50K bits/sec
82510 max Baud Rate	= 1 × 10/800 μs
(one byte FIFO)	= 12.5K bits/sec

4.2 Interrupt Handling

The 82510 has 16 different sources of interrupt, each of these sources, when set and enabled, will cause their respective block interrupt requests to go active. The block interrupt request, if enabled, will set the 82510's INT pin high, and will be reflected as a pending interrupt in the *General Interrupt Register (GIR)* if no other higher priority block is requesting service. If a higher priority block interrupt is also active at the same time, then the *General Interrupt Register* will reflect the higher priority request as the source of the 82510 interrupt. The lower priority interrupt will issue a new edge on the interrupt pin only after the higher priority interrupt is acknowledged and if no other priority block requests are present. Both the block interrupts and the individual sources within the blocks are maskable. The block interrupts are enabled through the *General Enable Register (GER)* which prevents masked bits in the *General Status Register (GSR)* from being decoded into the *General Interrupt Register.* This does not prevent the block request from being set in the *General Status Register*, it only prevents the masked GSR bits from being decoded into the *General Interrupt Register*, and thus generating any interrupts. The individual sources within the block are masked out via the corresponding interrupt enable register associated with the specific block (Rx Machine, Timing Unit and the Modem I/O module each have an Interrupt Enable register).

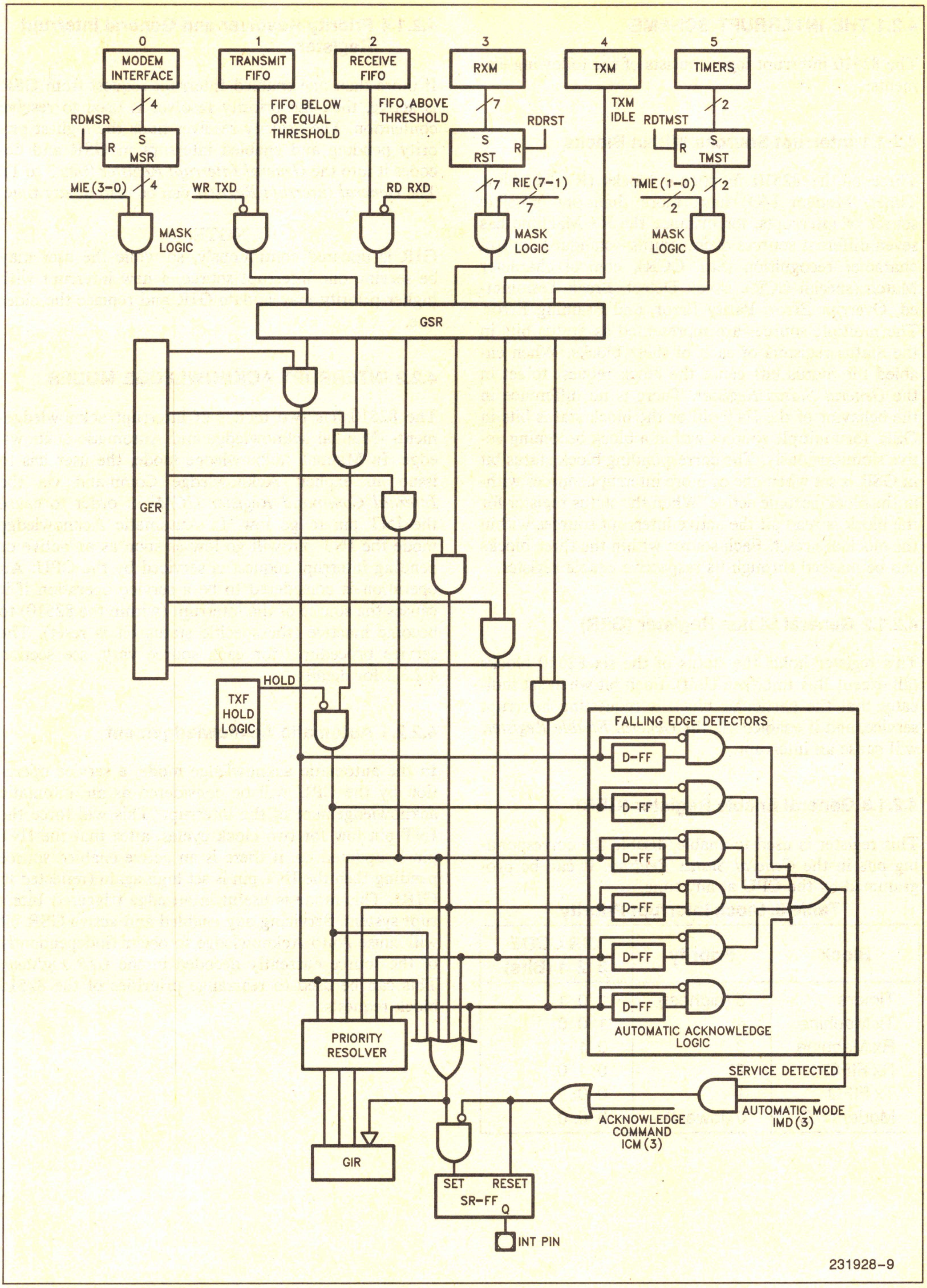


Figure 9. 82510's Interrupt Scheme

4.2.1 THE INTERRUPT SCHEME

The 82510 interrupt logic consists of the following elements:

4.2.1.1 Interrupt Sources Within Blocks

Three of the 82510 functional blocks (Rx Machine, Timer, Modem I/O) have more than one possible source of interrupts, for instance the Rx Machine has seven different sources of interrupts—standard control character recognition (Std. CCR), control character Match (special CCR), Break Detect, Break Terminated, Overrun Error, Parity Error, and Framing Error. The multiple sources are represented as Status bits in the Status registers of each of these blocks. When enabled the Status bits cause the block request to set in the *General Status Register.* There is no difference in the behavior of the INT pin or the block status bits in GSR, for multiple sources within a block becoming active simultaneously. The corresponding block status bit in GSR is set when one or more interrupt sources within the block become active. When the status register for the block is read all the active interrupt sources within the block are reset. Each source within the three blocks can be masked through its respective enable register.

4.2.1.2 General Status Register (GSR)

This register holds the status of the six 82510 blocks (all except Bus Interface Unit). Each bit when set indicates that the particular block is requesting interrupt service, and if enabled via the *General Enable Register,* will cause an interrupt.

4.2.1.3 General Enable Register (GER)

This register is used to enable/disable the corresponding bits in the *General Status Register.* It can be programmed by the CPU at any time.

Table 3. Block Interrupt Priority

Block	Priority	GIR CODE 3 2 1 (Bits)
Timers	5 (highest)	1 0 1
Tx Machine	4	1 0 0
Rx Machine	3	0 1 1
Rx FIFO	2	0 1 0
Tx FIFO	1	0 0 1
Modem I/O	0 (lowest)	0 0 0

4.2.1.4 Priority Resolver and General Interrupt Register

If more than one enabled Interrupt request from GSR is active, then the priority resolver is used to resolve contention. The priority resolver finds the highest priority pending and enabled interrupt in GSR and decodes it into the *General Interrupt Register* (bits 3 to 1). The *General Interrupt Register* can be read at any time.

NOTE:
GIR is updated continuously, so while the user may be serving one interrupt source, a new interrupt with higher priority may update GIR and replace the older one.

4.2.2 INTERRUPT ACKNOWLEDGE MODES

The 82510 has two modes of interrupt acknowledgement—Manual acknowledge and Automatic acknowledge. In Manual Acknowledge mode, the user has to issue an explicit Acknowledge Command via the *Internal Command Register (ICM)* in order to cause the INT pin to go low. In Automatic Acknowledge mode the INT pin will go low as soon as an active or pending interrupt request is serviced by the CPU. An operation is considered to be a service operation if it causes the source of the interrupt (within the 82510) to become inactive (the specific status bit is reset). The service procedures for each source vary, see section 4.2.3.2 for details.

4.2.2.1 Automatic Acknowledgement

In the automatic acknowledge mode, a service operation by the CPU will be considered as an automatic acknowledgement of the interrupt. This will force the INT pin low for two clock cycles, after that the INT pin is updated i.e. if there is an active enabled source pending then the INT pin is set high again (reflected in GIR). This mode is useful in an edge triggered Interrupt system. Servicing any enabled and active GSR bit will cause Auto Acknowledge to occur (independently of the source currently decoded in the *GIR register).* This can be used to rearrange priorities of the 82510 block requests.

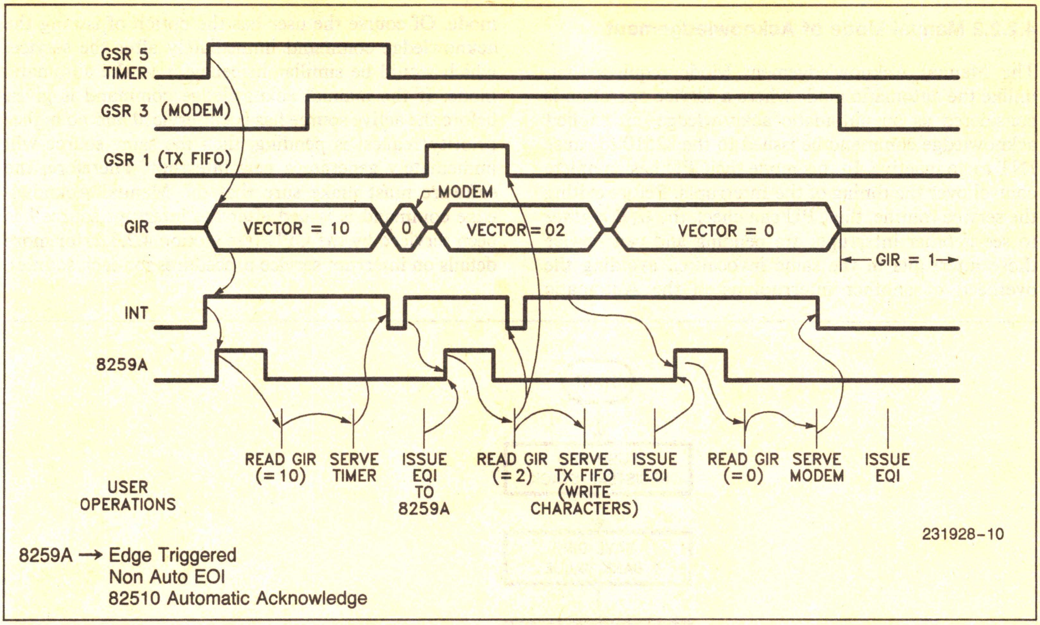


Figure 10. Automatic Acknowledge Mode Operation

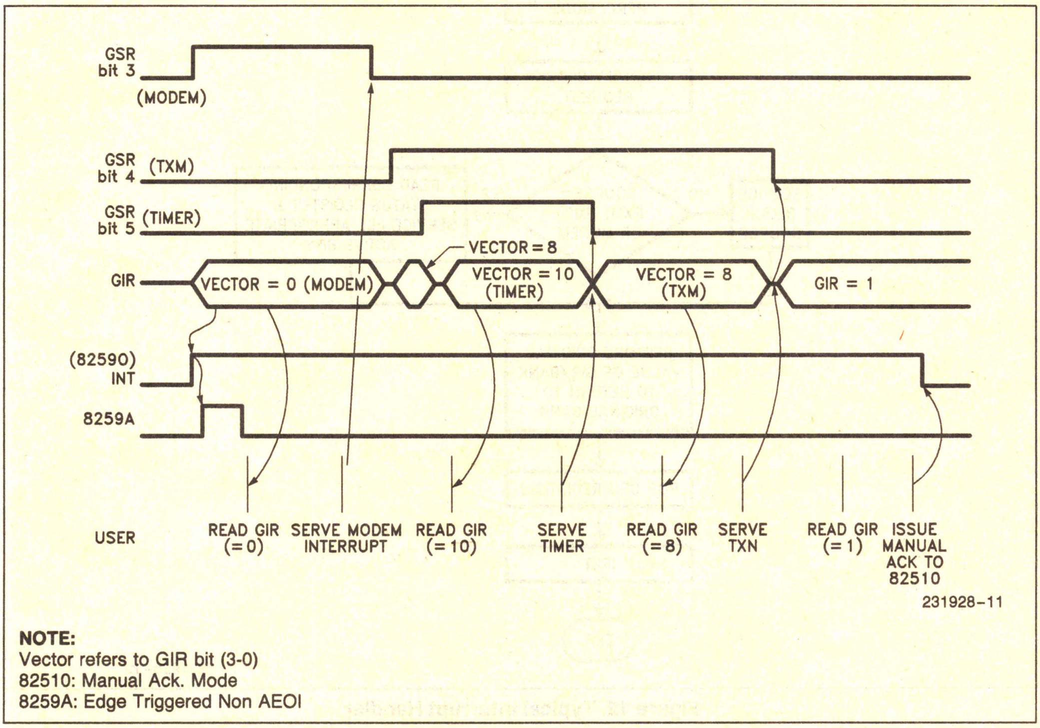


Figure 11. Manual Acknowledge Mode Operation

4.2.2.2 Manual Mode of Acknowledgement

The Manual Acknowledgement Mode requires that, unlike the automatic mode where a service operation is considered as an automatic acknowledge, an explicit acknowledge command be issued to the 82510 to cause INT to go inactive. In this mode the CPU has complete control over the timing of the Interrupts. Before exiting the service routine, the CPU can check the *GIR register* to see if other interrupts are pending and can service those interrupts in the same invocation, avoiding the overhead of another interrupt as in the Automatic mode. Of course the user has the option of issuing the acknowledge command immediately after the service, which would be similiar in behavior to the automatic mode. If the manual acknowledge command is given before the active source has been serviced and no higher priority request is pending, then the same source will immediately generate a new interrupt. Therefore, the software must make sure that the Manual Acknowledge command is issued after the interrupt source has been serviced by the CPU (see section 4.2.3.2. for more details on interrupt service procedures for each source).

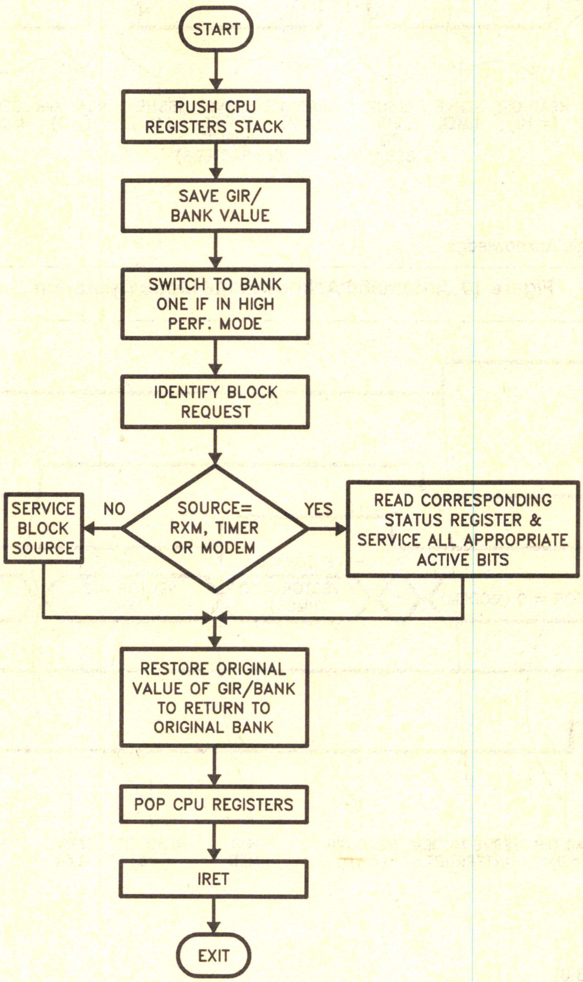


Figure 12. Typical Interrupt Handler

4.2.3 GENERAL INTERRUPT HANDLER

In general an interrupt handler for the 82510 must first identify the interrupt source within the 82510, transfer control to the appropriate service routine and then service the active source. The active source can be identified from two registers—*General Interrupt Register,* or *General Status Register.* The *GIR register* identifies the highest priority active block interrupt request. The *GSR register* identifies all active (pending or in service) Block Interrupt Requests. The typical operation of the 82510 interrupt handler is given in Figure 12. The two major issues of concern are the source identification and Control Transfer to the appropriate service routine.

Since the 82510 registers are divided into banks, and the interrupt handler may change register banks during service, it is best to save the bank being used by the main program and then do the interrupt processing. Upon completion of service, the original bank value is restored to the *GIR/Bank register.*

4.2.3.1 Source Identification

The 82510 has 16 interrupt sources, and the CPU must identify the source before performing any service. Although the procedure varies, the typical method would be to identify the block requesting service by reading

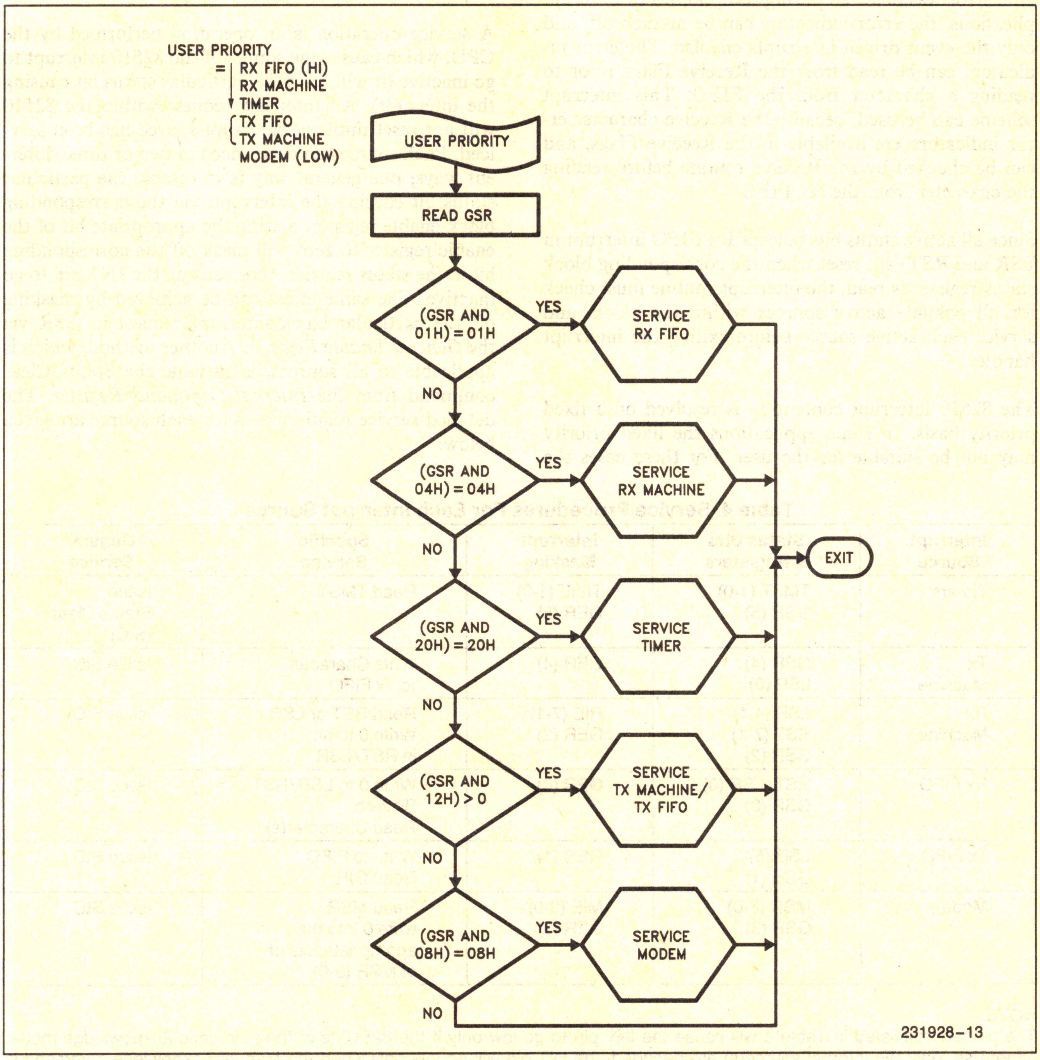


Figure 13. Bypassing the 82510 Fixed Interrupt Priority

GIR bits 3-1. If the source is either Tx Machine, Tx FIFO, or Rx FIFO, no further indentification is needed, the user can transfer control to the service routine (in most cases, only one Timer will be used, therefore the Timer Routine can also be directly invoked). All modem I/O interrupts can be handled via one routine as all the modem interrupt sources are supplementary to the modem handshaking function. The Rx Machine, however, has two different types of interrupt sources, event indications (CCR/Address recognition CCR/Address Match, Break Detect, Break Terminate, and Overrun Error), and error indications (Parity Error, Framing Error, these error indications do not refer to any particular character, they just indicate that the specific error was detected during reception). For most applications, the error indicators can be masked off, and only the event driven interrupts enabled. The error indicators can be read from the Receive Flags prior to reading a character from the FIFO. This interrupt scheme can be used, because the Receive character error indicators are available in the Receive Flags, and can be checked by the Receive routine before reading the character from the Rx FIFO.

Since all active status bits (except Rx FIFO interrupt in LSR and RST) are reset when the corresponding block status register is read, the interrupt routine must check for all possible active sources within the block, and service each active source before exiting the interrupt handler.

The 82510 interrupt contention is resolved on a fixed priority basis. In some applications the fixed priority may not be suitable for the user. For these cases the user can bypass the 82510's priority resolution by using the *General Status Register* (rather than GIR) to determine the block interrupt sources requesting service. Each source is checked in order of user priority and serviced when identified (There will be no problem with using this algorithm in auto acknowledge mode because the INT pin will go low as soon as a pending and enabled interrupt request goes low). The user will be trading some service latency time for additional source identification time, this algorithm's efficiency will improve as the number of block sources to verify is reduced. See Figure 13 for the algorithm.

4.2.3.2 Interrupt Service

A service operation is an operation performed by the CPU, which causes the source of the 82510 interrupt to go inactive (it will reset the particular status bit causing the interrupt). An interrupt request within the 82510 will not reset until the interrupt source has been serviced. Each source can be serviced in two or three different ways; one general way is to disable the particular status bit causing the interrupt, via the corresponding block enable register. Setting the appropriate bit of the enable register to zero will mask off the corresponding bit in the status register, thus causing the INT pin to go inactive. The same effect can be achieved by masking off the particular block interrupt request in GSR via the *General Enable Register.* Another method, which is applicable to all sources, is to issue the Status Clear command from the *Internal Command Register.* The detailed service requirements for each source are given below:

Table 4. Service Procedures For Each Interrupt Source

Interrupt Source	Status Bits & Registers	Interrupt Masking	Specific Service	General Service
Timers	TMST (1-0) GSR (5)	TMIE (1-0) GER (5)	Read TMST	Issue Status Clear (StC)
Tx Machine	GSR (4) LSR (6)	GER (4)	Write Character to Tx FIFO	Issue StC
Rx Machine	LSR (4-1) RST (7-1) GSR (2)	RIE (7-1) GER (2)	Read RST or LSR Write 0 to bit in RST/LSR	Issue StC
Rx FIFO	RST/LSR (0) GSR (0)	GER (0)	Write 0 to LSR/RST Bit zero. Read Character(s)	Issue StC
Tx FIFO	LSR (5) GSR (1)	GER (1)	Write to FIFO Read GIR	Issue StC
Modem	MSR (3-0) GSR (3)	MIE (3-0) GER (3)	Read MSR write 0 into the appropriate bits of of MSR (3-0)	Issue StC

NOTE:
The procedures listed in Table 4 will cause the INT pin to go low only if the 82510 is in the automatic acknowledge mode. Otherwise, only the internal source(s) are decoded, the INT pin will go low only when the Manual Acknowlege command is issued.

4.3 Polling

The 82510 can be used in a polling mode by using the *General Status Register* to determine the status of the various 82510 blocks, this is useful when the software must manage all the blocks at once. If the software is dedicated to performing one function at a time, then the specific status registers for the block can be used, e.g. if the software is only going to be Transmitting, it can monitor the Tx FIFO level by polling the *FIFO Level Register,* and write data whenever the Tx FIFO level decreases. Reception of data can be done in the same manner.

5.0 SOFTWARE CONSIDERATIONS

5.1 Configuration

The 82510 must be configured for the appropriate modes before it can be used to transmit or receive data. Configuration is done via read and write registers, each functional block (except for BIU) has a configuration register. Typically the configuration is done once after start up, however, the FIFO thresholds and the interrupt masks can be reconfigured dynamically. If the 82510 configuration is not known at start up it is best to bring the device to a known state by issuing a software reset command (ICM register, bank one). At this point all block interrupts are masked out in GER and all configuration and status registers have default values. The bank register is pointing to bank zero. The 82510 can now be configured as follows:

1. If BRG A is being issued as a baud rate generator then load the baud rate count into BAL and BAH registers.

2. Configure the character attributes in LCR register (Parity, Stop Bit Length, and Character Length).

(Note if interrupts are being used, steps 1 and 2 can also be done at the end, since the user will have to return to bank zero to set the interrupt masks in GER)

3. Load ACR0 register with the appropriate Control or Address character (if using the Control Character Match or Address Match capability of the 82510).

4. Switch to Bank two.

(In this Bank the configuration can be done in any order)

5. Configure the Receive and Transmit FIFO thresholds if using different thresholds than the default).

6. Configure the Transmit Mode Register for the Stop Bit length, modem control, and if using echo or 9 bit length or software parity, configure the appropriate bits of the register. The default mode of the modem control is Manual, if using the FIFO then the automatic mode would be most useful).

7. Configure the Rx FIFO depth, interrupt acknowledge mode, μlan or normal mode and echo modes in IMD register.

8. Load ACR1 if necessary

9. Enable Rx Machine Interrupts as necessary via RIE.

10. Configure RMD for CCR, DPLL operation, Sampling Window, and start bit.

11. Switch to Bank 3.

12. Configure CLCF register for Tx and Rx clocks and or Sources

13. Configure BACF register for BRG A mode and source.

14. Load BBL and BBH if BRGB is being used (as either a BRG or a Timer).

15. Configure BBCF register if necessary.

16. If reconfiguration of the modem pin is necessary then program the PMD register.

17. Enable any modem interrupt sources, if required, via MIE register.

18. Enable Timer interrupts, if necessary, via TMIE.

19. If using interrupts

then

i) Switch to Bank zero.

Disable Interrupts at CPU (either by masking the request at the interrupt controller or executing the CLI instruction).

ii) Enable the appropriate 82510 Block interrupts by setting bits in the GER register. (CPU interrupts can now be reenabled, but it is recommended to switch banks before enabling the CPU interrupts).

NOTE:
At this stage it is best to leave the TxM and Tx FIFO interrupt disabled. See section 6.3 Transmit Operation for details)

20. Switch to Bank One. Load Transmit Flags if using 9-bit characters, or 8051 9-bit mode or software parity. If using interrupts CPU interrupts can now be enabled.

Bank One is used for general operation, the 82510 can now be used to transmit or receive characters.

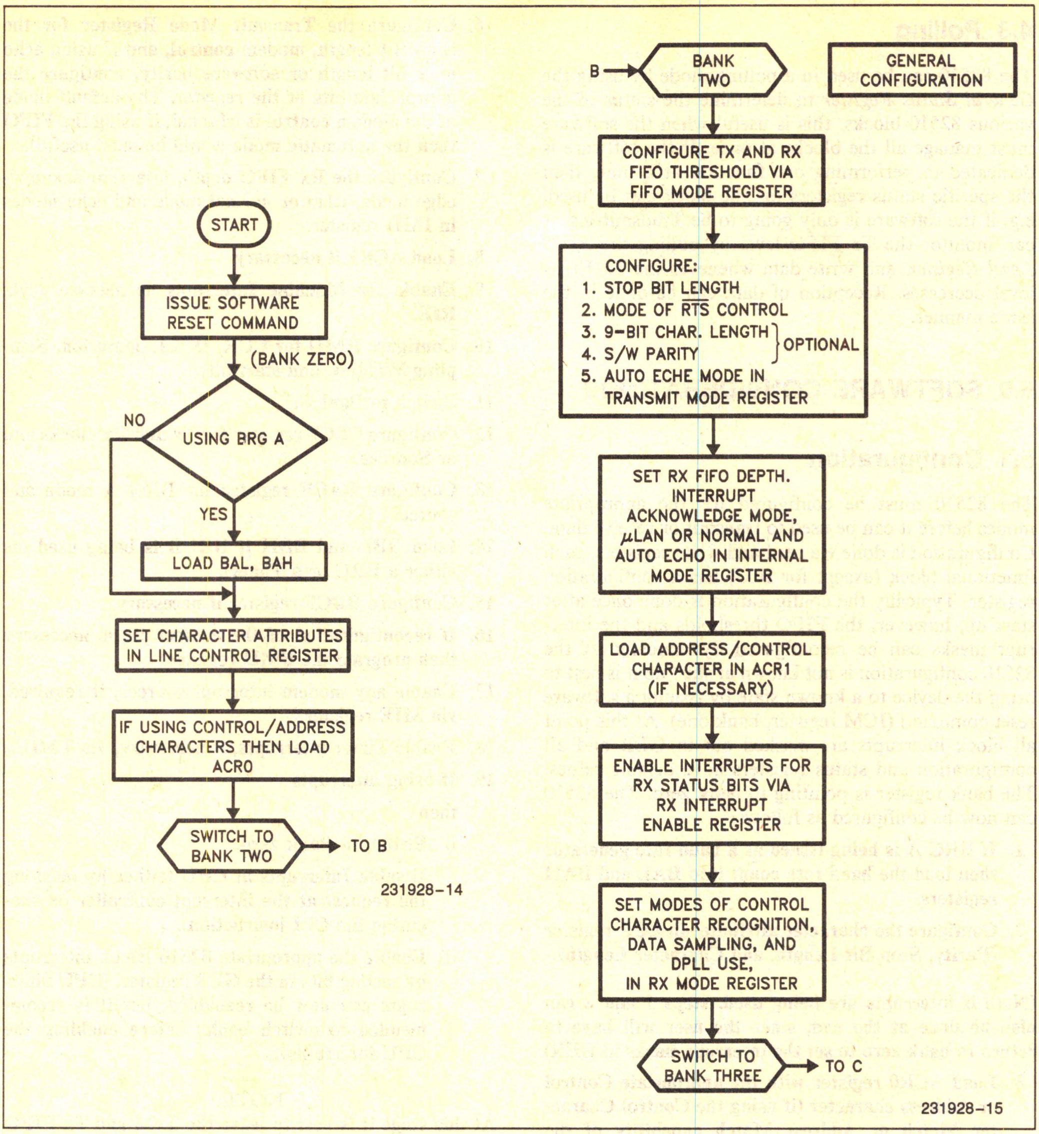


Figure 14. Configuration Flow Chart

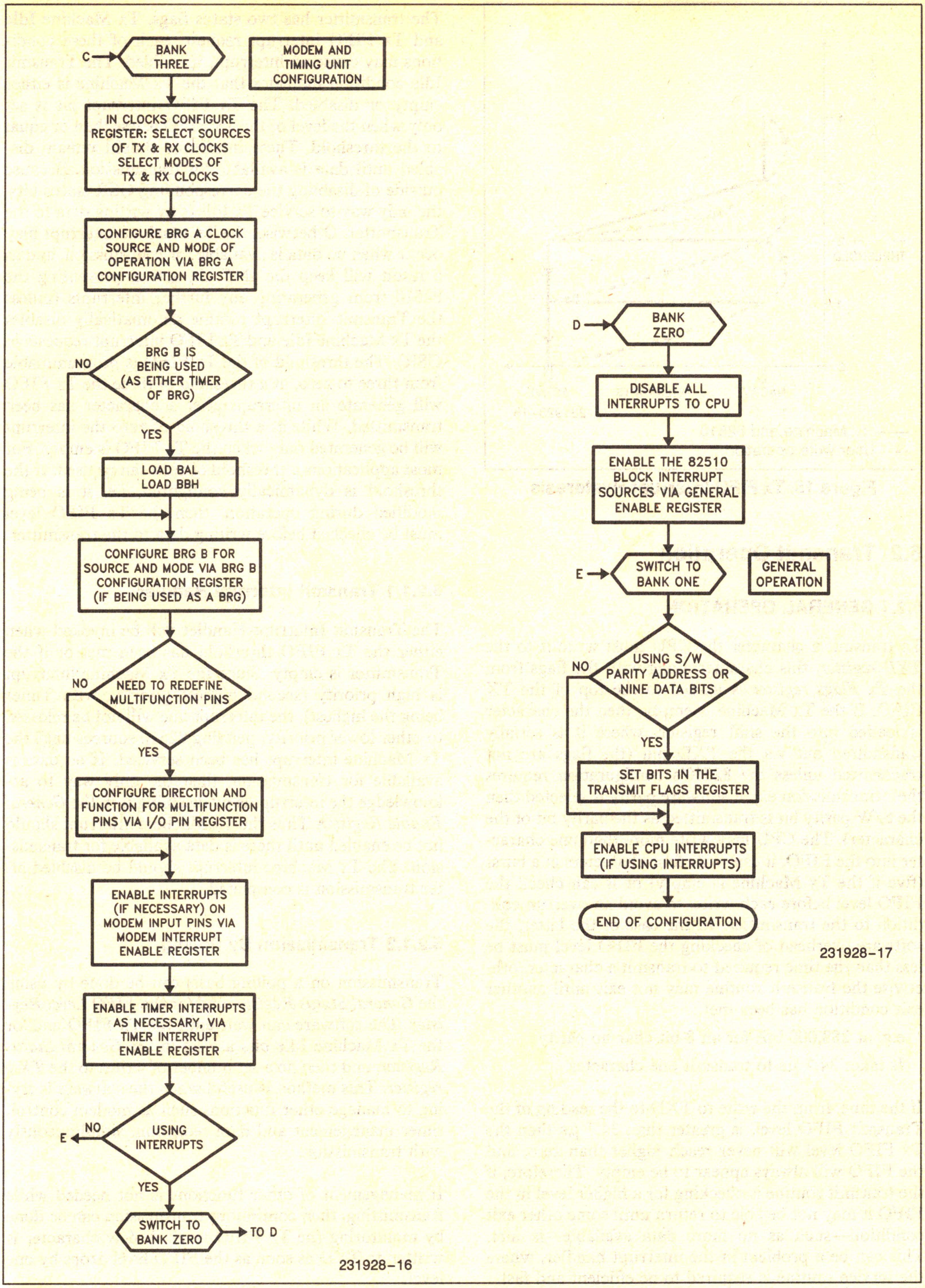


Figure 14. Configuration Flow Chart (Continued)

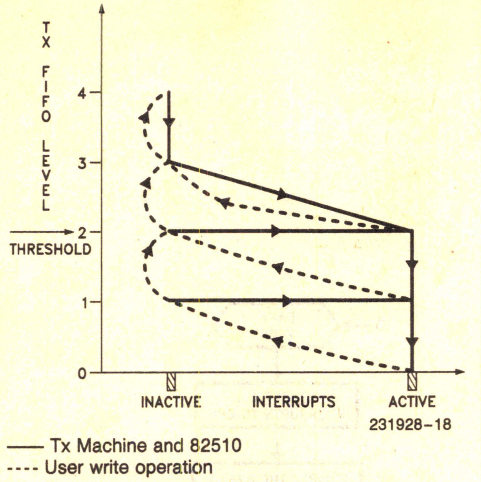


Figure 15. Tx FIFO Interrupt Hysteresis

5.2 Transmit Operation

5.2.1 GENERAL OPERATION

To transmit a character the CPU must write it to the *TXD register*, this character along with the flags from the *Tx Flags register* is loaded to the top of the TX FIFO. If the Tx Machine is empty, then the character is loaded into the shift register, where it is serially transmitted out via the TXD pin (the flags are not transmitted unless the 82510's configuration requires their transmission e.g. if software parity is selected then the S/W parity bit is transmitted as the parity bit of the character). The CPU may write more than one character into the FIFO, it can write four characters in a burst (five if the Tx Machine is empty) or it can check the FIFO level before each write, to avoid an overrun condition to the transmitter. In the case of the latter, the software overhead of checking the FIFO level must be less than the time required to transmit a character, otherwise the transmit routine may not exit until another exit condition has been met.

e.g. at 288,000 bps for an 8-bit char no parity

It takes 34.7 μs to transmit one character.

If the time, from the write to TXD to the reading of the Transmit FIFO level, is greater than 34.7 μs then the Tx FIFO level will never reach higher than zero, and the FIFO will always appear to be empty. Therefore, if the transmit routine is checking for a higher level in the FIFO it may not be able to return until some other exit condition—such as no more data available—is met. This can be a problem in the interrupt handler, where the service routine is required to be efficient and fast.

The transmitter has two status flags. Tx Machine Idle and Tx FIFO interrupt request, each of these conditions may cause an interrupt, if enabled. The Transmit Idle condition indicates that the Tx Machine is either empty or disabled. The Tx FIFO interrupt bit is set only when the level of the Tx FIFO is less than or equal to the threshold. These interrupts should remain disabled until data is available for transmission. Because outside of disabling the corresponding GSR status bits, the only way to service Tx Idle is by writing data to the Transmitter. Otherwise, the Tx Machine interrupt may occur when no data is available for transmission, and as a result will keep the INT pin active, preventing the 82510 from generating any further interrupts (unless the Transmit Interrupt routine automatically disables the Tx Machine Idle and Tx FIFO interrupt requests in GSR). The threshold of the Tx FIFO is programmable from three to zero, at a threshold of three the Tx FIFO will generate an interrupt after a character has been transmitted. While at a threshold of zero the interrupt will be generated only when the Tx FIFO is empty. For most applications a threshold of zero can be used. If the threshold is dynamically configured, i.e. it is being modified during operation, then the Tx FIFO level must be checked before writing data to the transmitter.

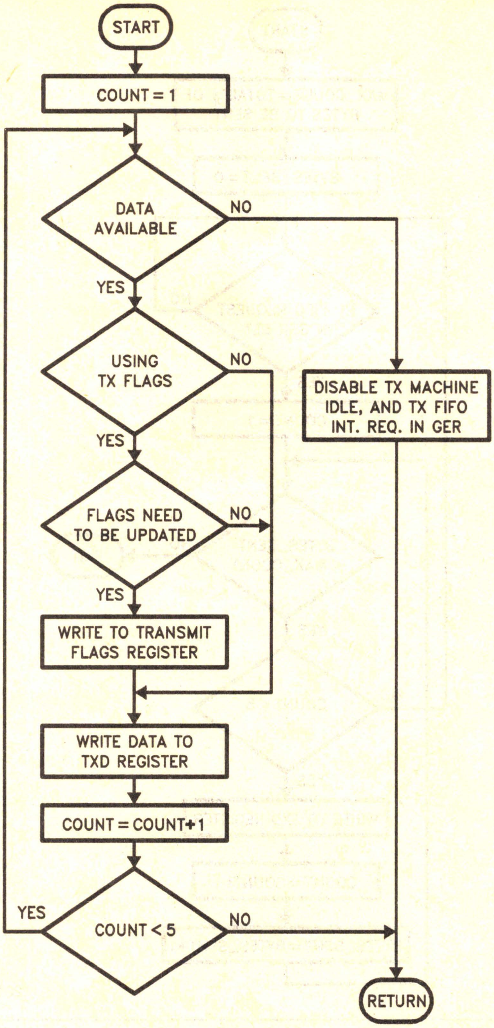
5.2.1.1 Transmit Interrupt Handler

The Transmit Interrupt Handler will be invoked when either the Tx FIFO threshold has been met or if the Transmitter is empty. Since the Tx Machine interrupt is high priority (second highest priority, with Timer being the highest), the interrupt line will not be released to other lower priority, pending 82510 sources until the Tx Machine interrupt has been serviced. If no data is available for transmission, then the only way to acknowledge the interrupt is by disabling it in the *General Enable Register*. Thus the Tx Machine interrupt should not be enabled until there is data available for transmission. The Tx Machine interrupt should be disabled after transmission is completed.

5.2.1.2 Transmission By Polling

Transmission on a polling basis can be done by using the *General Status Register* and/or the *FIFO Level Register*. The software can wait until the Tx FIFO and/or the Tx Machine Idle bits are set in the *General Status Register*, and then do a set number of writes to the *TXD register*. This method is useful when the software is trying to manage other functions such as modem control, timer management and data reception, simultaneously with transmission.

If management of other functions is not needed while transmitting, then continuous transmission can be done by monitoring the Tx FIFO level. A new character is written to TXD as soon as the FIFO level drops by one level.



Tx FIFO Threshold = 0

231928-19

NOTE:
TxM Idle and Tx FIFO Empty interrupts are enabled by the Main Program, when data transmission is required.

Figure 16. 16 Tx Interrupt Handler Flow Chart

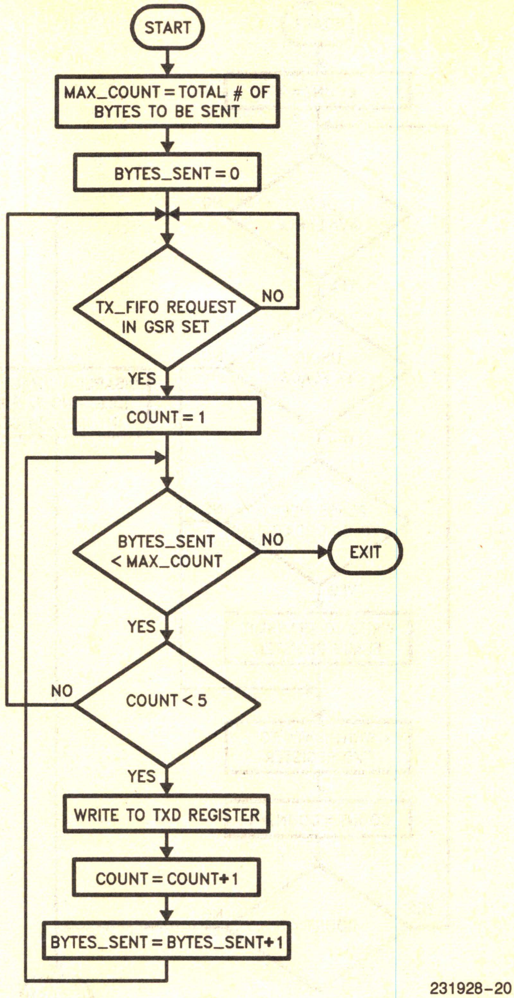


Figure 17. Using GSR for Polling

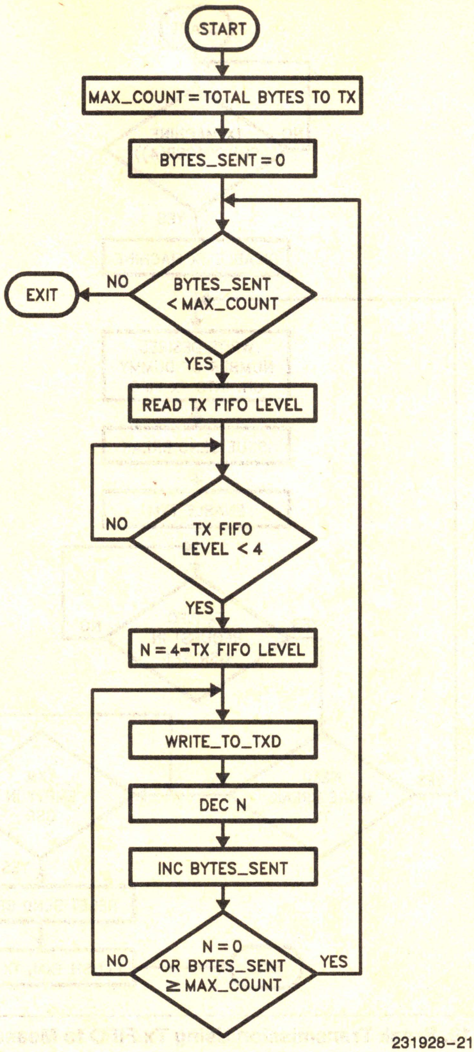


Figure 18. Data Transmission by Monitoring FIFO Level

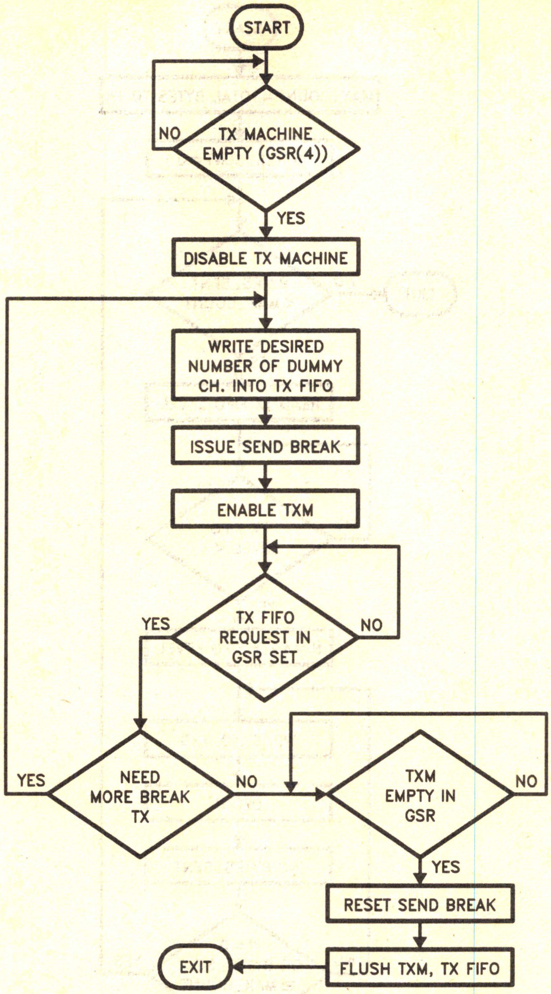


Figure 19. Break Transmission Using Tx FIFO to Measure Break Length

5.2.1.3 Break Transmission

The 82510 will transmit a break when bit six of the *Line Control Register* is set high. This will cause the TXD pin to be held at Mark for one or more character time. The Tx FIFO can be used to program a variable length break, see Figure 19 for details. If the break command is issued in the midst of character transmission the TXD pin will go low, but the transmitter will not be disabled. The characters from the Tx FIFO will be shifted out on to the Tx Machine and lost. To prevent the erroneous transmission of data, The CPU must make sure the Transmitter is empty or disabled before issuing the Send Break command.

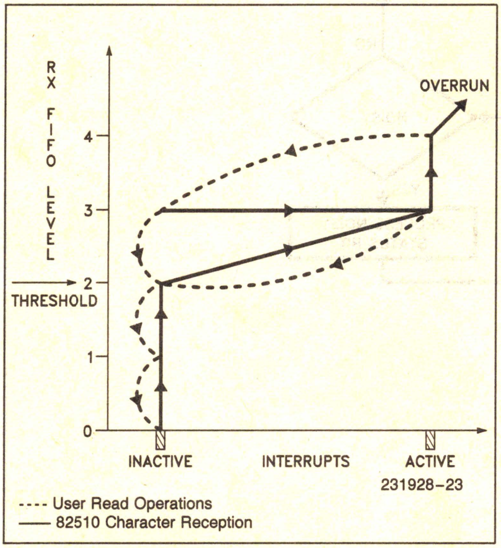


Figure 20. Rx FIFO Hysteresis

5.3 Data Reception

The receiver provides the 82510 with three types of information:

a) Data characters received

b) Rx Flags for each data character

c) Status information on events within the Rx Machine.

The Rx FIFO interrupt request goes active when the Rx FIFO level is greater than the threshold, if the interrupt for this bit is enabled then it will generate an interrupt to the CPU. This is a request for the CPU to read characters from the 82510. Each character on the Rx FIFO has flags associated with it, all of these flags are generated by the Rx Machine during reception of the character. These flags provide information on the integrity of the character, e.g. whether the character was received OK, or if there were any errors. The receiver status is provided via the *Receive Status Register (RST),* which provides information on events occurring within the Rx Machine, since the last time RST was read. The information may or may not apply to the current character being read from the *RXD register.* The CPU may read one or more characters from the Rx FIFO. After each read, if the FIFO contains more than a single character, a new character is loaded into the *RXD register* and the flags for that character are placed into the *RXF register.* The software can check for the Rx character OK bit in the flags to make sure that the character was received without any problems.

5.3.1 RECEIVE INTERRUPT HANDLER

The Receiver will generate two types of interrupts, Rx FIFO interrupt and Rx Machine Interrupt. The Rx FIFO interrupt requires that the CPU read data characters from the Rx FIFO. If the Rx Machine interrupts are disabled then the CPU should also check for errors in the character before moving it to a valid buffer. The interrupts generated by the Rx Machine can be divided into two categories—occurrence of errors during reception of data (parity error, framing error, overrun error), or the occurrence of certain events (Control/Address character received, Break detected, Break Terminated). For typical applications, the error status of each received character can be checked via the *Receive Flags,* and the events can be handled via interrupts.

5.3.2 RECEIVING DATA BY POLLING

To receive data through polling, the 82510 can use the *General Status or the Receive Status Registers* to check for the Rx FIFO request. If the Receive routine does not generate time outs or modem pin transitions, then the data can also be received by monitoring the Rx FIFO level in the *FIFO Level Register.* The implementation using GSR would be useful in applications where the software routine must monitor the timer for time outs or the modem pins for change in status. The example polling routine illustrates the use of the *FIFO Level Register* in receiving data. It waits for the Rx FIFO request before beginning data reception. The procedure Rx_Data_Poll will receive the number of characters requested in Char_count and place them in the Receive buffer.

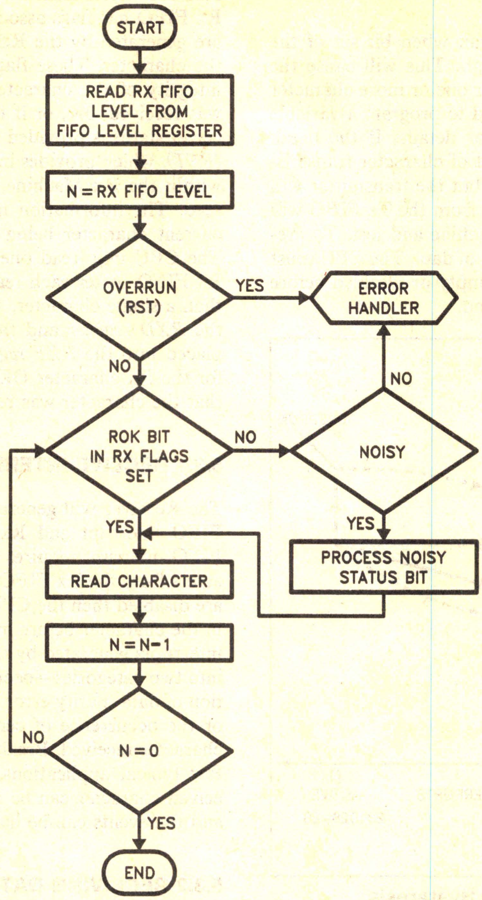


Figure 20. Rx FIFO Interrupt Handler

```
#define base 0x3F8;      /* base address of 82510 */
#define buff__size 128;

Rx__Data__Poll (Char__count, Rxbuffer)
int Char__count;          /* Total # of bytes to be received */
char *Rxbuffer [buffsize];
{
int count = 0;
int status, IvI, Rok;

While (((status = (Inp(base+7) & 0x05)) == 0x01) /* If Rx FIFO Req in GSR set */
{                                                 /* Assume in bank one */
   /* If Rx FIFO is not empty */
   While ((IvI = ((Inp (base+4) & 0x70)/0x10)0&&(count < (Char__count))
   {
      /* If Character Received OK */
      if (((Rok = (Inp (base+1) & 0x60)) == 0x40)
      {
         Rxbuffer [count] = Inp (base);
         ++count;
      {
   {
{
{
```

Figure 21. Example Polling Routine

5.4.3 CONTROL CHARACTER HANDLING

The 82510 has two modes of control character recognition. It can recognize either standard ASCII or standard EBCDIC control characters, or it can recognize a match with two user programmed control (or Address Characters in MCS-51 9-bit mode, for Automatic Wake up) characters. Each mode generates an interrupt through the *Receive Status Register.* The *Receive Flags* also indicate whether the character being read is a control character. The usage of CCR depends on the maximum number of possible control characters that can be received at any one time. Applications such as Terminal Drivers, which have no more than two control characters outstanding, such as XON and Ctl-C, or XOFF and Ctl-C, can use just the Control Character Match mode by programming the registers ACR0 and ACR1. If the CPU needs to process text on a line by line basis, the standard Control Character recognition capability can be used to determine when an end of line has occurred e.g. a whole line has been received when a Carriage Return (CR) or Line Feed (LF) is received by the UART.

Implementation of a character oriented asynchronous file transfer protocol can be done using both standard and specific Control Character Recognition. In such protocols most control characters such as Start of Header (SOH), can only be received during certain states, these characters can be received via Standard Control Character Recognition. A few Control Characters (e.g. abort) can be received at any stage of communication, these can be received by using the Control Character Matching capabilities of the 82510.

5.3.3 BREAK RECEPTION

The 82510 has two status indications of break reception, Break Detect indicates that a break has been detected on the RXD pin. Break Terminated indicates that the Break previously detected on the RXD line has terminated and normal Data reception can resume. Each of these status bits can generate an interrupt request through the Rx Machine Interrupt request. Normal consequence of break is to abort the data reception or to introduce a line idle delay in the middle of data reception. In the case of the former, the Break Detect interrupt can be used to reset the 82510 Receive Machine and the Rx routine flags; in the case of the latter, the break terminated interrupt can be used to filter out the break characters and resume normal reception. Each break character is identified by a break flag in the *Rx Flags Register* (the CCR flag, Framing error, and CCR Match flag also may become active when a break character is received) and is loaded onto the Rx FIFO as a NULL character. If break continues even after the Rx FIFO is full, then an overrun error will occur but no further break characters will be loaded on to the Rx FIFO. The user can also measure the length of the break character stream by using the Timer.

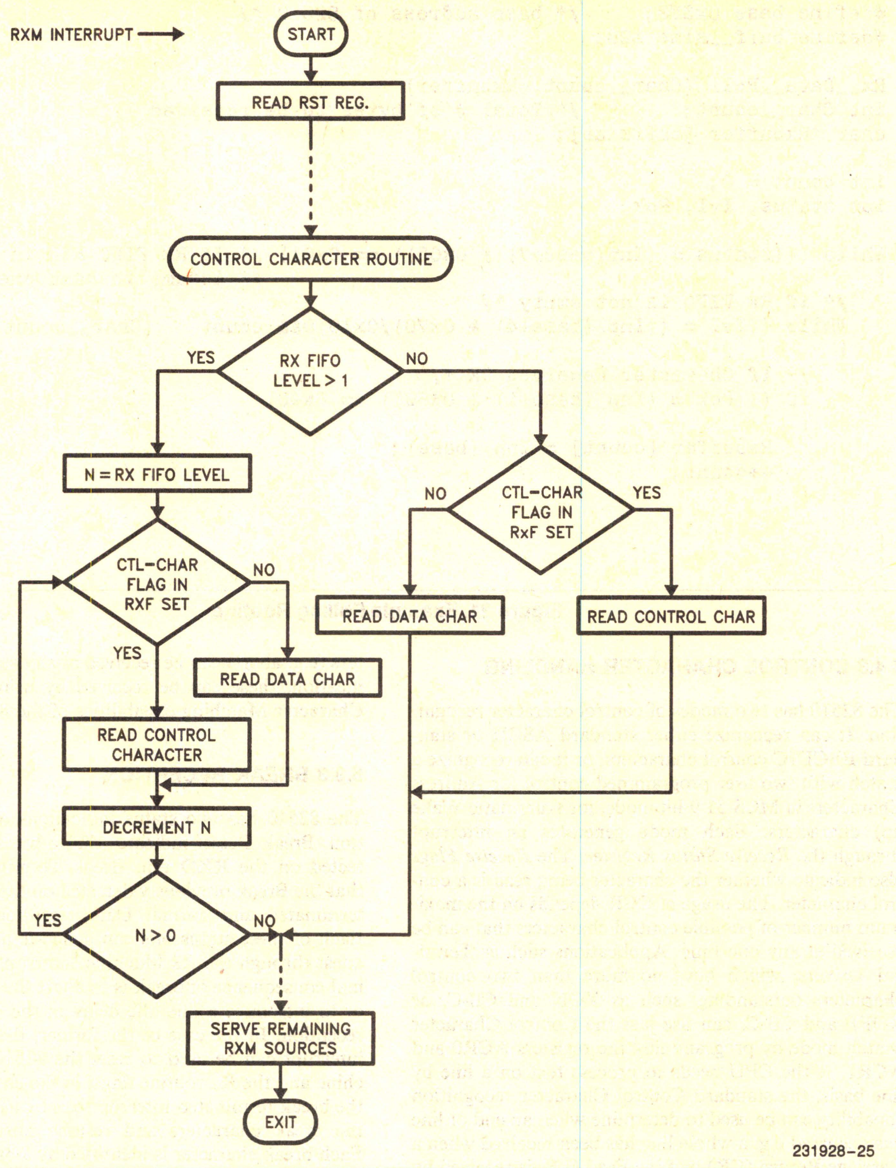


Figure 22. Handling Control Character Interrupts

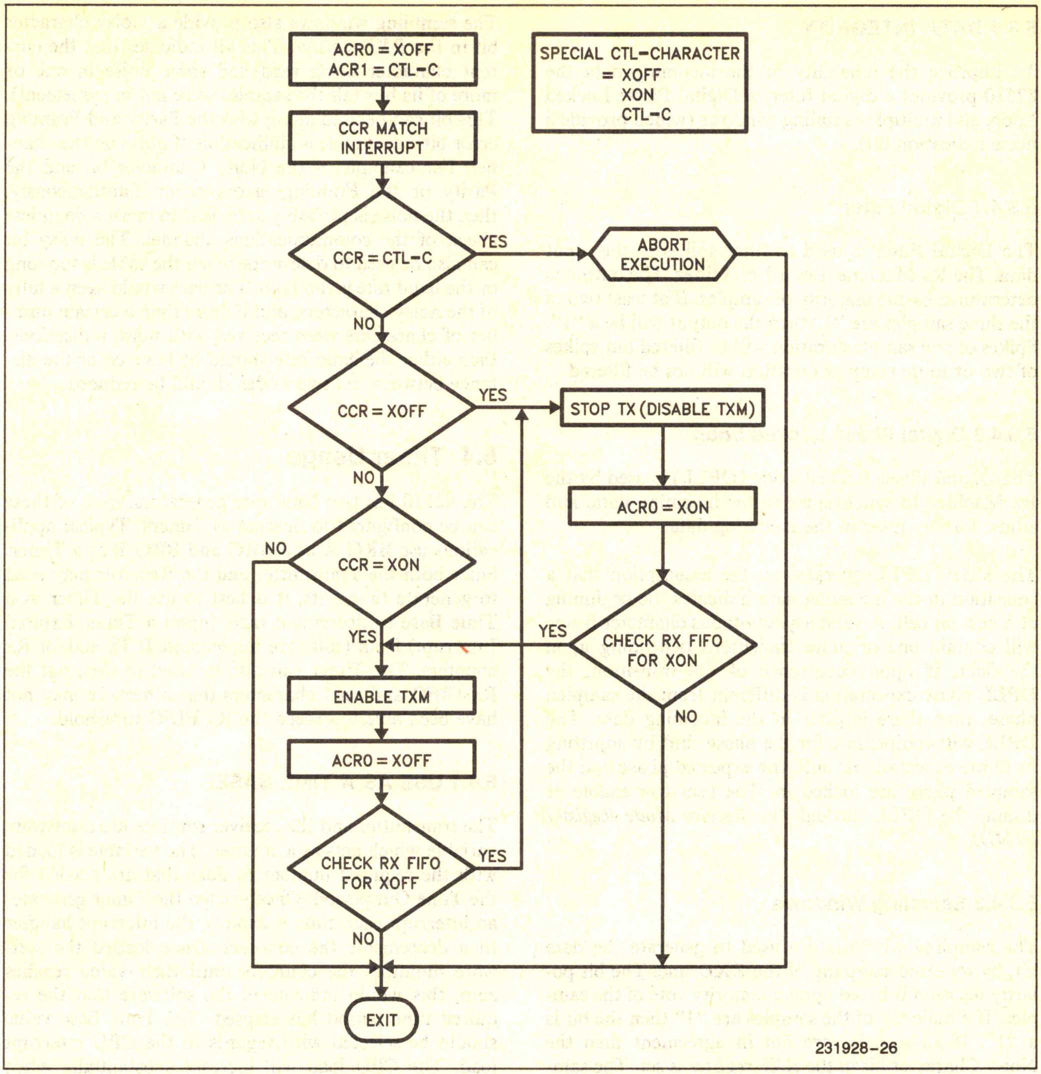


Figure 23. Using Control Character Match in Terminal Ports

5.3.4 DATA INTEGRITY

To improve the reliability of the incoming data the 82510 provides a digital filter, a Digital Phase Locked Loop, and multiple sampling windows (which provide a noise indication bit).

5.3.4.1 Digital Filter

The Digital Filter is used to filter spikes in the input data. The Rx Machine uses a 2 of 3 filter. The output is determined by the majority of samples. If at least two of the three samples are "1" then the output will be a "1". Spikes of one sample duration will be filtered but spikes of two or more samples duration will not be filtered.

5.3.4.2 Digital Phase Locked Loop

The Digital Phase Locked Loop (DPLL) is used by the Rx Machine to synchronize to the incoming data, and adjust for any jitter in the incoming data.

The 82510 DPLL operates on the assumption that a transition in the incoming data indicates the beginning of a new bit cell. A valid asynchronous character frame will contain one or more transitions depending upon the data. If upon occurrence of the transition, the DPLL phase expectation is different from the sampled phase, then there is jitter in the incoming data. The DPLL will compensate for the phase shift by adjusting its phase expectations, until the expected phase and the sampled phase are locked in. The user can enable or disable the DPLL through the *Receive Mode Register (RMD).*

5.3.4.3 Sampling Windows

The sampling windows are used to generate the data bit, by repeated sampling of the RXD line. The bit polarity decision is based upon a majority vote of the samples. If a majority of the samples are "1" then the bit is a "1". If all samples are not in agreement then the Noisy Character bit in the *RXF register* is set. The sampling windows are programmable for either 3 of 16 or 7 of 16. The 3/16 mode improves the jitter tolerance of the medium. While the 7/16 window improves the impulse noise tolerance of the channel.

The sampling windows also provide a Noisy character bit in the *RXF register.* This bit indicates that the current character being read had some noise in one or more of its bits (all the samples were not in agreement). This bit can be used along with the Parity and Framing error bits to provide an indication of noise on the channel. For example, if the Noisy Character bit and the Parity or the Framing errors occur simultaneously, then the noise is probably sufficient to merit a complete check of the communications channel. The noisy bit can also be used to determine when the cable is too long or the baud rate is too high. The user would keep a tally of the noisy characters, and if more than a certain number of characters were received with noise indications, then either the baud rate should be lowered or the distance between the two nodes should be reduced.

5.4 Timer Usage

The 82510 has two baud rate generators, each of these can be configured to operate as Timers. Typical applications use BRG A as a BRG and BRG B as a Timer. Since both the Transmitter and the Receiver may need to generate time outs, it is best to use the Timer as a Time Base to decrement ticks (upon a Timer Expired Interrupt) from (software implemented) Tx and/or Rx counters. The Timer can also be used to time out the Rx FIFO and read characters that otherwise may not have been able to exceed the Rx FIFO threshold.

5.4.1 USE AS A TIME BASE

The transmitter and the receiver routines use a software variable which acts as a counter. The variable is loaded with the required number of ticks that are needed for the Time Out period. Once started the Timer generates an interrupt each time it expires, the interrupt handler then decrements the counters. Once loaded the software monitors the counters until their value reaches zero, this would indicate to the software that the required time period has elapsed. The Time Base value should be selected with regards to the CPU interrupt load. The CPU load will increase substantially when the Timer is used as a Time Base, therefore using the Timer in this mode at very high baud rates may cause character overruns. A time base of 5 or 1 ms is probably the most useful. An additional benefit of the Time Base is that it can support more than two counters if required.

BRG-B is used as Timer.
BRG-A is used as BRG.
TB Ex bit in TMST Enabled.
Tx_Timer_Count contains count for Transmitter.
Rx_Timer_Count contains count for Receiver.

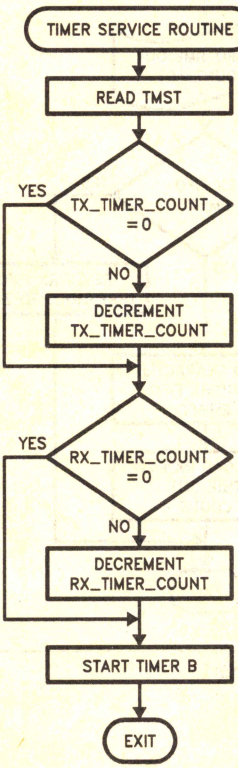


Figure 24. Timer use as Time Base for Transmit and Receive

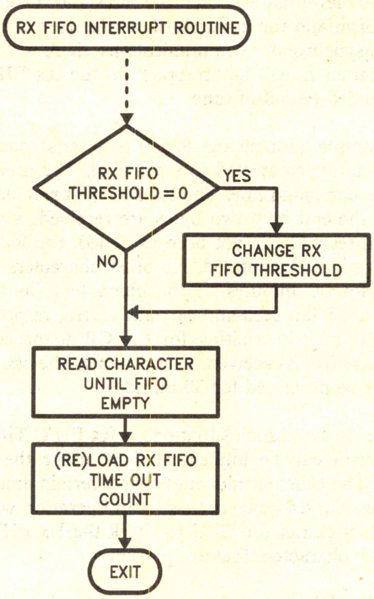
5.4.2 USE FOR RX FIFO TIME OUT

In the 82510, Rx FIFO interrupts will occur only after the FIFO level has exceeded the threshold. Due to this mechanism and the nonuniform arrival rate of characters in asynchronous communications, there is a chance that characters will be "trapped" in the Rx FIFO for an extended period of time.

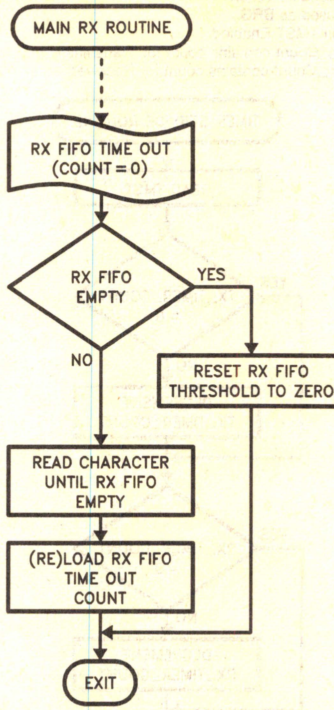
For example, assume the 82510 is a serial port on a system and is connected to a terminal. The user is entering a command line. The Rx FIFO Threshold = 3, and at the end only two bytes are received. Since the FIFO threshold has not been exceeded, the Rx FIFO interrupt is not generated. No other characters are received for 30 minutes, if the characters (in the Rx FIFO) are a line feed and carriage return, respectively, the CPU may be waiting for the CR to process the characters it has received. Consequently the characters will not be processed for 30 minutes.

In order to avoid such situations, a Rx FIFO Time Out mechanism can be implemented by using the 82510 Timer. The time out indicates that a certain amount of time has elapsed since the last read operation was performed. It causes the CPU to check the Rx FIFO and read any characters that are present.

In applications where the character reception occurs in a spurious manner (the exact number of characters cannot be guaranteed), the Rx FIFO Time Out is the only way to prevent characters from being trapped. The time out period is measured from the last read operation, every read operation resets the Rx FIFO Timer. To synchronize with the beginning of the data reception, initially the Rx FIFO threshold is set to zero. After the first character has been received, the threshold is adjusted to the desired value. When a Rx FIFO time out occurs and no data is available, the threshold is reset to zero. In error free data transmission, the beginning of data transmission is signaled by the reception of a control character, such as SOH or STX, the Rx FIFO time out mechanism should be triggered to the reception of these control characters.



231928-28



231928-29

Figure 25. Rx FIFO Time Out Flow Chart

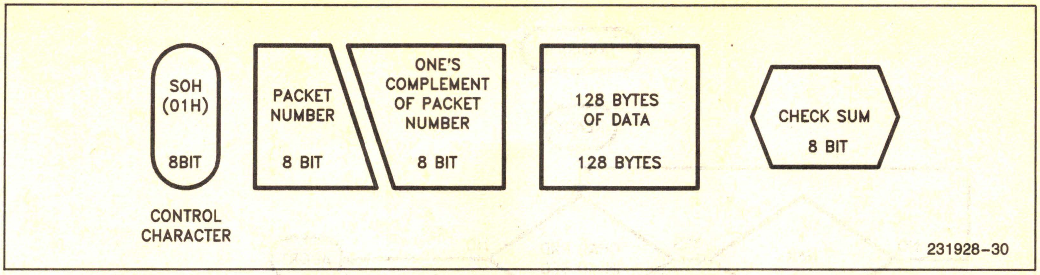


Figure 26. Packet Structure of XMODEM

6.0 82510 IMPLEMENTATION OF XMODEM

The 82510 XMODEM implementation is a file transfer program for the 82510 based on the XMODEM protocol. The software runs on the PC AT on a especially designed adapter board (the adapter board design is shown in Figure 33). The software uses most of the 82510 features including the baud rate generator, Timer, Control Character Recognition and FIFOs. The software uses an interrupt driven implementation, written in both assembly and C languages.

6.1 XMODEM Protocol

XMODEM is a popular error free data transfer protocol for asynchronous communications. Data is transferred in fixed length 128 byte packets, each packet has a checksum for error checking. The packets are delineated by control characters, which act as flags between the Receiver and the Transmitter. There are four control characters, SOH, EOT, ACK, and NAK. SOH indicates the Start of a Packet, EOT indicates the End Of Transmission; ACK and NAK are positive or negative acknowledgements of the packet respectively. The packet structure and protocol flow of XMODEM is provided in the figures given below.

6.2 Software

Interrupts are used to transmit and receive data. The software is implemented as two independent finite state machines—Transmit State Machine and Receive State Machine. Each state machine is triggered by external events such as user commands and data or Control Character reception. The state machines communicate with the 82510 interrupt service routines through software flags. The overall structure of the main routine is given in Figure 31. The major modules of the software are given in the hierarchy Chart, Figure 34, which lists the different modules in order.

The interface between the main program and the interrupt service routine is done through global flags. The interrupt handler services four sources—Transmit, Timer, Receive, and Control Characters. Each of the interrupt sources communicates with each of the state machines through the global flags. The state machines keep track of their individual states through state variables. The interface between the individual states within a state machine is done through state flags. The state machine diagrams are given in Figure 29 and Figure 30.

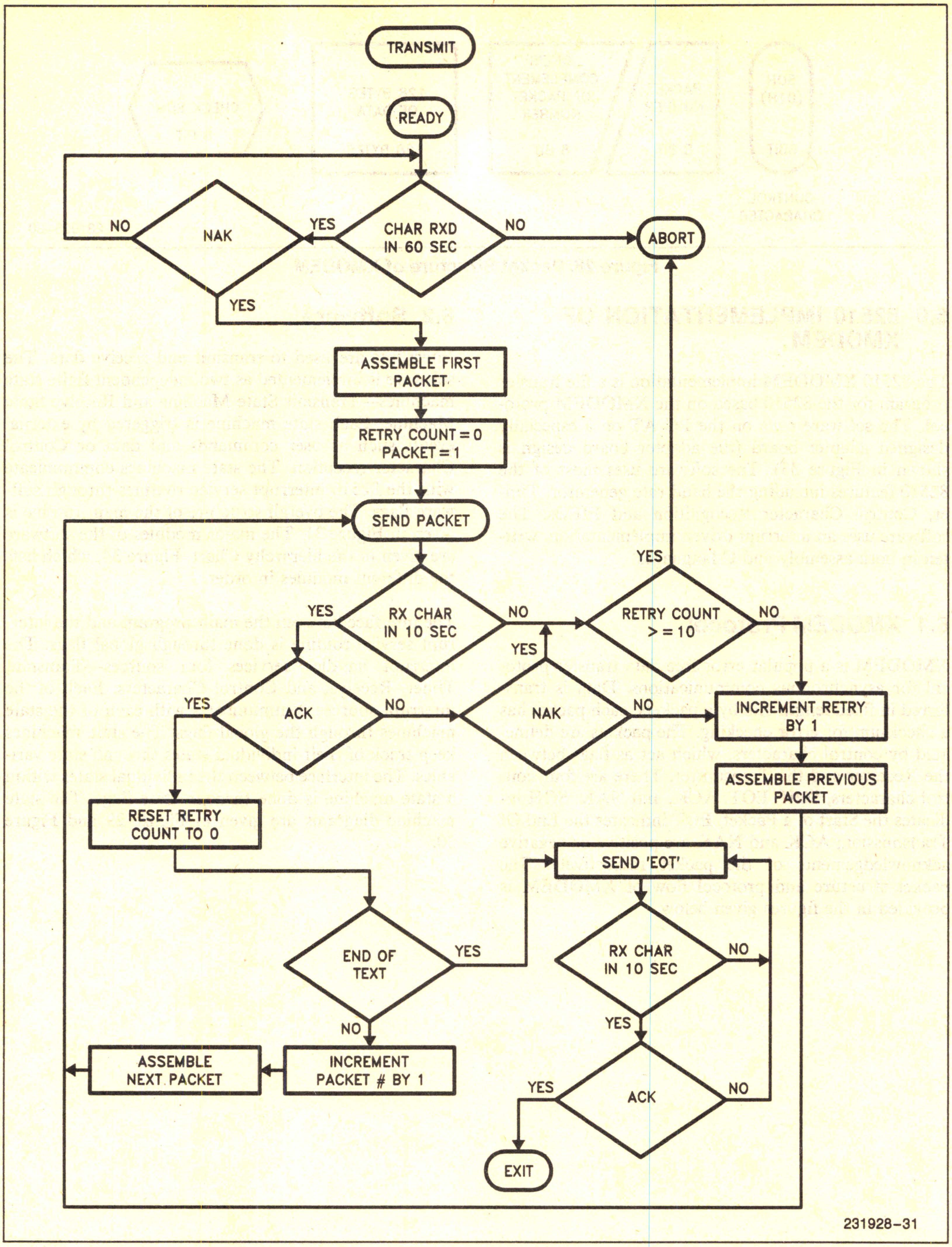


Figure 27. Protocol Flow for Transmit Side of XMODEM

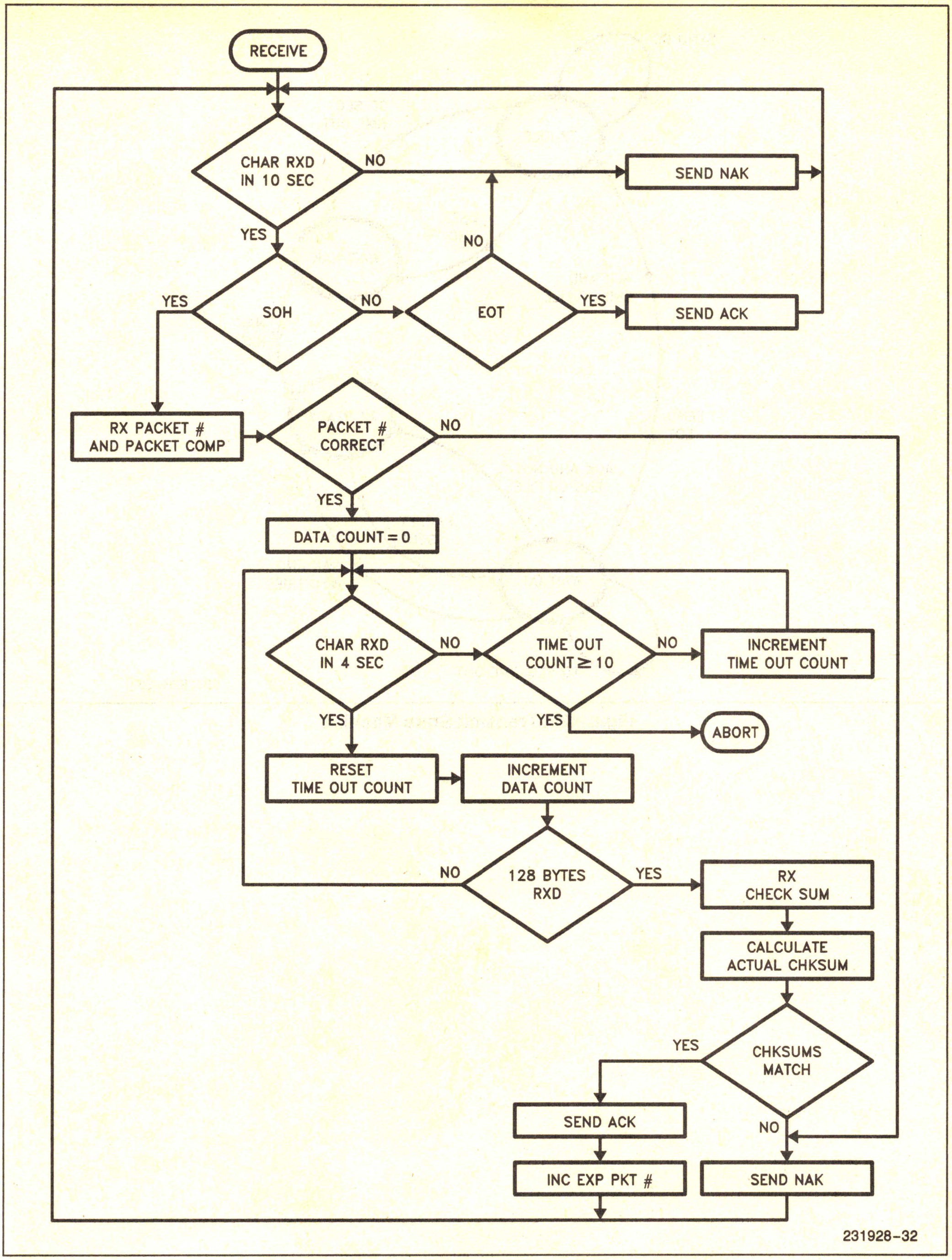


Figure 28. Protocol Flow for Receive Side of XMODEM

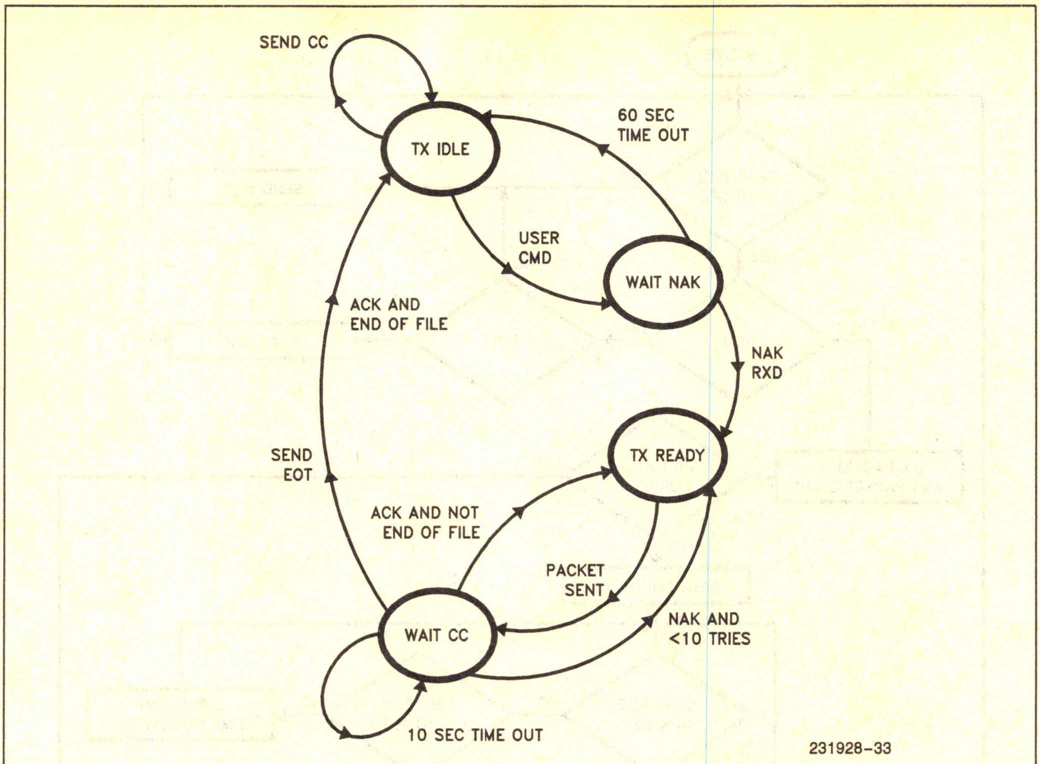


Figure 29. Transmit State Machine

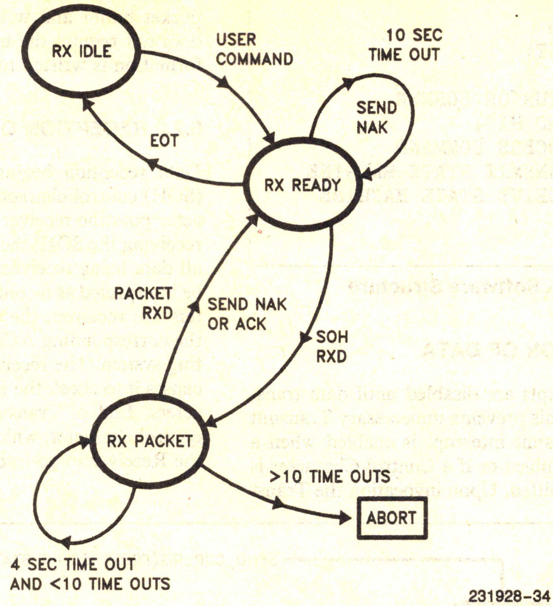


Figure 30. Rx State Machine

```
START
Initialization
WHILE (NOT QUIT)
{
   UPDATE STATUS ON SCREEN
   IF (KEYBOARD HIT)
      THEN PROCESS COMMAND
   PROCESS TRANSMIT STATE MACHINE
   PROCESS RECEIVE STATE MACHINE
}
END
```

Figure 31. Software Structure

6.2.1 TRANSMISSION OF DATA

The Transmit interrupts are disabled until data transmission is required, this prevents unnecessary Transmit interrupts. The Transmit interrupt is enabled when a packet has been assembled or if a Control Character is required to be transmitted. Upon invocation the Transmit interrupt service routine reads characters from the packet buffer and writes it to the Tx FIFO. Since it does not require the use of the Transmit Flags, no information is written to the *TXF register.*

6.2.2 RECEPTION OF DATA

Data reception begins only after a Start of Header (SOH) control character is received. This control character puts the receiver in a data reception mode. After receiving the SOH, the CCR interrupt is disabled (since all data being received now is transparent and can not be interpreted as a control character). After 132 characters are received, the CCR interrupt is reenabled and the corresponding ACK or NAK sent to the Transmitting system. The receiver has a time out feature, which causes it to check the Rx FIFO for any remaining characters. End of Transmission is indicated by an EOT control character, which causes the file to be closed and the Receiver to go into the Idle state.

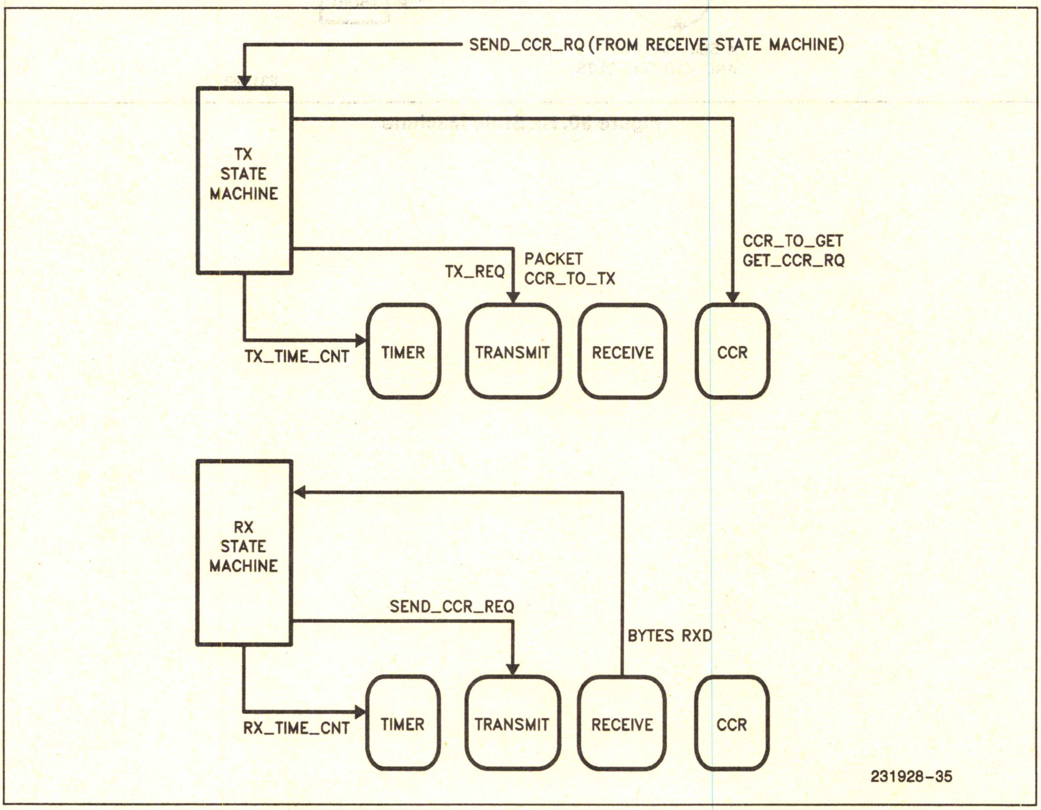


Figure 32. Using Flags for Communications with Interrupt Routine

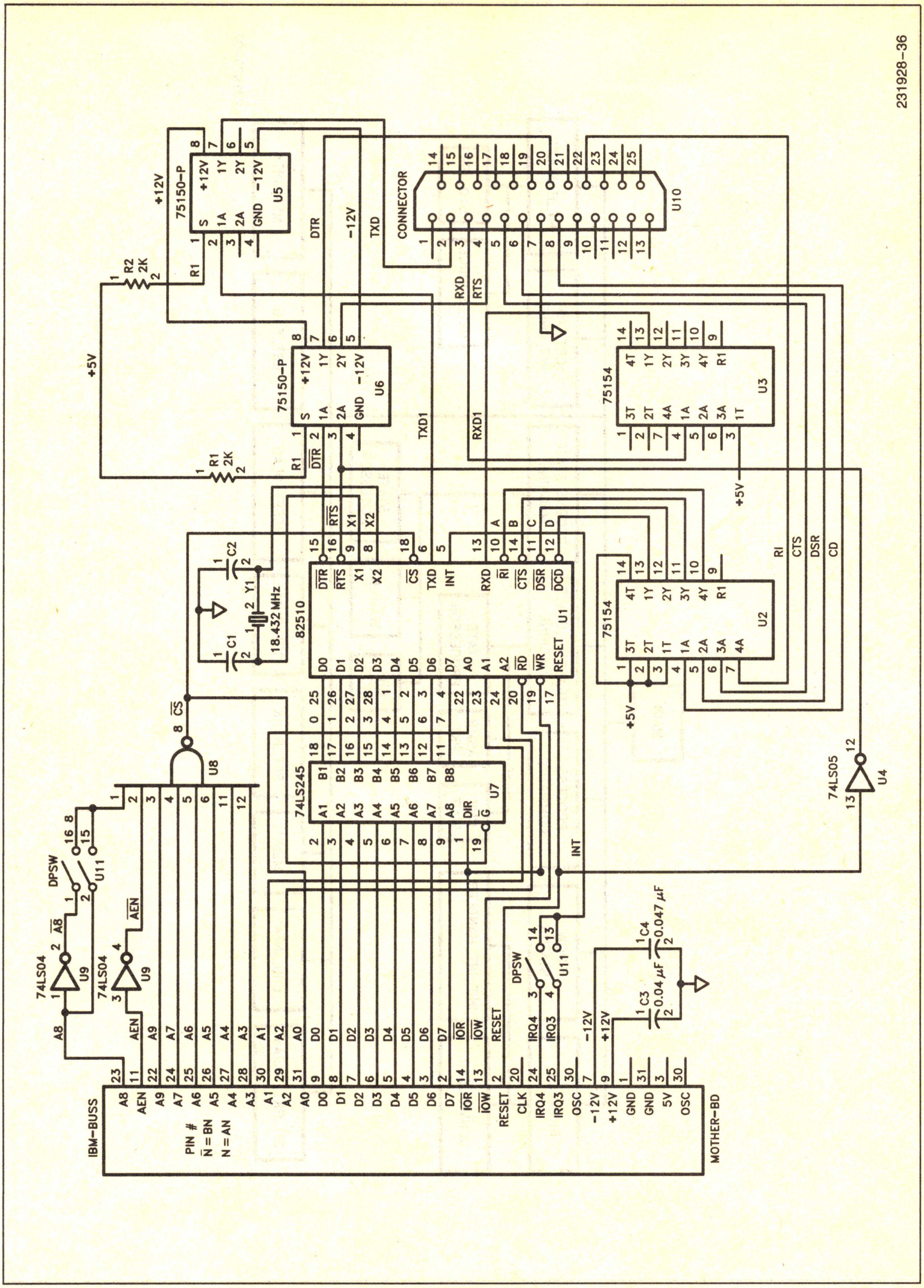


Figure 33. PC AT Adapter

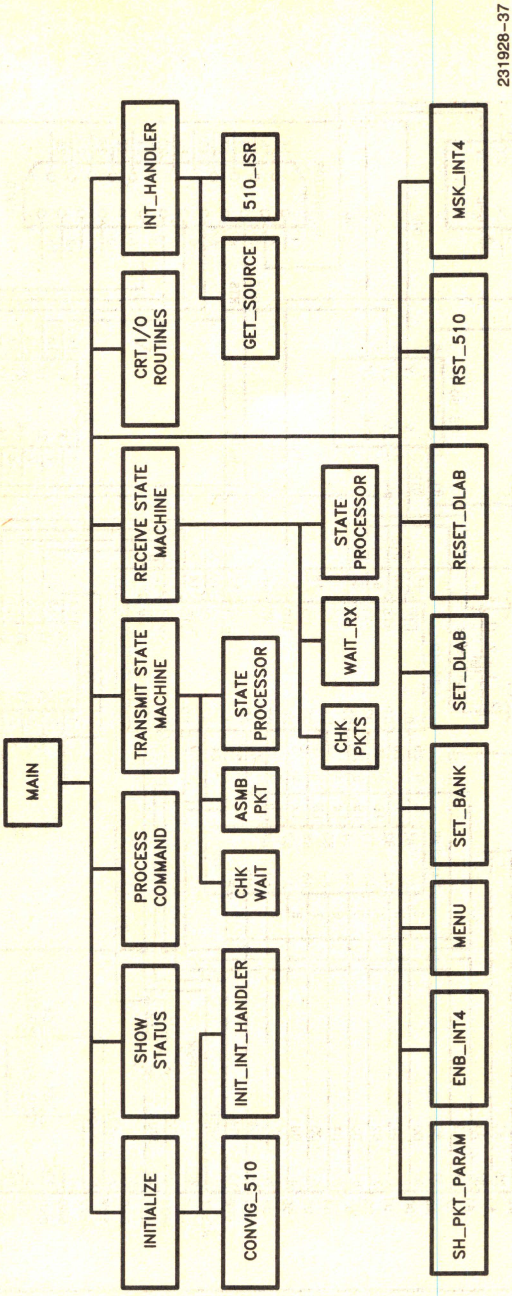


Figure 34. Hierarchy Chart

6.3 Software Listings

```
PAGE  1        MAIN PROGRAM    ftp.c   82510 XMODEM

#include "C:\ftp\ftp.def"
#include "C:\lc\fcntl.h"
#include "C:\lc\stdlib.h"
#include "C:\lc\stdio.h"
/************************************************************/
/***                                                 *****/
/***     SEPTEMBER 1986                              *****/
/***                                                 *****/
/***     82510 XMODEM IMPLEMENTATION                 *****/
/************************************************************/
int     eof =false;             /* end of file flag */
int     rxpk =0;
int     txrflg;
int     rxrflg;
int     exp_pkt_num = 1;        /* next packet number expected by receiver */
int     pkst;
int     rxtocnt;                /* Time Out counter for receiver */
int     quit =false;
int     key = 0;
int     sohcnt =0;              /* # of SOH characters received */
int     rxfcnt =0;              /* # of Rx FIFO Interrupts */
int     ccrcnt =0;              /* # of Ctl-Char. Interrupts */
int     tx_state =tx_idle;      /* Transmitter State Variable */
int     rx_state = rx_idle;     /* Receiver State Variable */
int     tx_cmd = inactive;      /* Indicates a Valid Tx Command was given */
int     rx_cmd = inactive;      /* Indicates a valid Rx Command was issued */

/* File to be Transmitted */
char    tx_file_name[40] = "                                        ";

/* File to be Received */
char    rx_file_name[40] = "                                        ";

int     send_ccr_req = inactive;        /* Flag - Request to Tx Ctl-Char */
int     intvec =0;                      /* contains the GIR vector */
int     l;
char    txdata [132];                   /* Tx Buffer */
char    rxbuf [128];                    /* Rx Buffer */
char    rxdata [131];
char    rx_f_buf [32000];               /* Rx File Stored in this buffer */

/******************************/
/*** tx state variables *****/
/******************************/
int     tx_indx;                        /* Pointer to the next character in the
                                           buffer */

struct  packet    {
        char            head;
        char            pack_num;
        char            pack_cmpl;
        char            buffer [128];
        char            chksm;
     };

struct packet rxpack, txpack;
```

231928-38

82510 XMODEM Implementation

```
PAGE  2        MAIN PROGRAM    ftp.c  82510 XMODEM


/**********************************************************/
/****          tx State machine and interrupt      ******/
/****          handler  flags                      ******/
/**********************************************************/

/** txm and tx fifo   **/
int       tx_req =0;        /* Flag - indicates a request for transmission to
                               82510 Interrupt Handler */
int       ccr_to_tx = 0;    /* Actual Ctl-char to Transmit */
int       tx_byte_cnt =0;   /* Total # of Bytes Transmitted */
int       pkts_sent =0;     /* # of Packets sent */

/** Timer    **/
int       tx_time_cnt =0;   /* Transmitter Timer Counter */

/** CCR **/
int       get_ccr_rq =0;    /* Flag - Request to Receive Ctl-character */
int       ccr_to_get  =0;   /* Received Ctl-char value */


/****************************************************************/
/****                                                   *****/
/****          RX STATE VARIABLES                       *****/
/****                                                   *****/
/****                                                   *****/
/****************************************************************/

char      pk_chksm;         /* Calculated Chksum */
int       eot_cnt =0;       /* # of EOTs Received */
int       bad_pkt_cnt;      /* # of Bad Packets Received */

/*********************************************************/
/****          rx state machine and Interrupt     *****/
/****          handler flags                      *****/
/*********************************************************/

/** rx fifo **/
int       rx_byte_cnt =0;        /* # of Bytes Received */

/** CCR **/

int       ctl_rxd_flg =0;        /* Flag-Indicating that a Ctl-Char. has been
                                    receivd*/
int       rx_ctl_chr=0;          /* Actual Ctl-char received */

/** Timer    **/
int       rx_time_cnt =0;        /* Receive Timer Count */
```

231928-39

82510 XMODEM Implementation (Continued)

```
PAGE  3        MAIN PROGRAM    ftp.c  82510 XMODEM


/******************************************/
/**           MAIN ROUTINE             ***/
/******************************************/

main ()
{
int       q,txfl,rxfl;
int       rval,v =0;
int       cmd = 0;
int       wn_status =0;
int       ecode = 0;
FILE      *fp;
FILE      *rxfp;
int       rwst;
int       wcst;
int       retx_cnt =0;                /* Retransmit count */
int       tocnt =0;                   /* Time Out Count */
int       tx_secs,rx_secs;
int       i,s, lpcnt = 0;

     CLR ();                          /* Clear Screen */
     MV_CURS (so_r,so_c);             /* Sign On Message */
     printf (s1);
     init ();                         /* Initialize 82510 and Variables */
     MENU ();                         /* Print Menu */
     enbint4 ();                      /* Enable Interrupts in 8259A */
     outp (ip00,eoi);                 /* issue EOI     */
     outp ((bpa+3),0x22);             /* start timer B */
     lpcnt =0;                        /* Keeps Track of # of Loops */

/****************************************************/
/***             main while loop               ****/
/****************************************************/
     while (quit==false)
     {

     /**************************************/
     /** display protocol parameters     **/
     /**************************************/

       ++ lpcnt;
       mv_curs (4,30);
       printf ("loop # = %u",lpcnt);
       mv_curs (4,50);
       printf ("rx int. cnt = %u",rxfcnt);
       mv_curs (5,50);
       printf ("ccr int cnt = %u",ccrcnt);
       mv_curs (4,1);
       printf ("interrupt vector = %u \n", intvec);
       q = inp (bpa+4);
       txfl = q & 0x07;
       mv_curs (5,1);
       printf ("TX FIFO = %u ",txfl);
       q = inp (bpa+4);
       rxfl = q & 0x70;
       mv_curs (6,1);
       printf ("RX FIFO = %u \n",rxfl/16);
       mv_curs (6,50);
       printf ("SOH count = %3u",sohcnt);
```

231928–40

82510 XMODEM Implementation (Continued)

```
PAGE  4        MAIN PROGRAM   ftp.c  82510 XMODEM

       mv_curs (7,1);
       printf ("bytes received %3u",rx_byte_cnt);
       mv_curs (7,30);
       printf ("Bytes Sent = %3u",tx_byte_cnt);
       mv_curs (7,50);
       printf  ("EOT count %3u", eot_cnt);
       mv_curs (5,30);
       printf ("pkts rxd = %3u",(exp_pkt_num-1));
       mv_curs (6,30);
       printf ("pkts sent = %3u", pkts_sent);
       tx_secs = tx_time_cnt/200;
       rx_secs = rx_time_cnt/200;
       open_wind (3,1,"Tx Timer");
       printf (" = %2u secs",tx_secs);
       open_wind (3,50,"Rx Timer");
       printf (" = %2u secs",rx_secs);
       mv_curs (8,1);
       printf ("Bad Packets Rxd = %3u",bad_pkt_cnt);
       mv_curs(8,30);
       printf ("# of ReTx packets = %3u",retx_cnt);

   /* If Command Issued then process the Command */
       if ((key =kbhit()) > 0)
          quit = process_cmd ();

       else
       {

         /************************************************************/
         /*****          Process Tx STATE MACHINE            *****/
         /*****          revision 0                          *****/
         /*****                                              *****/
         /************************************************************/

         switch (tx_state) {
             case tx_idle:

         /************************************************************/
         /****                                              *****/
         /****  TRANSMITTER IDLE STATE                      *****/
         /****                                              *****/
         /****               Checks for a Send Ctl-Char.    *****/
         /****               Checks for the Transmit Command *****/
         /****                                              *****/
         /****                                              *****/
         /************************************************************/


                 /* If Control Character to be Transmitted Then Transmit the
                    Control Character by setting the Tx_req Flag and enabling
                    the TxM and Tx FIFO interrupts */

                 if ((send_ccr_req ==  active) && (!tx_req))
                 {
                     tx_req =ctl_chr;
                     tx_i_enb ();
                     while ( tx_req>0);
                     tx_i_dis ();
                     send_ccr_req=inactive;
                 }
```

231928–41

82510 XMODEM Implementation (Continued)

```
PAGE  5        MAIN PROGRAM    ftp.c  82510 XMODEM

            /* If the Transmit Command is issued then Wait for a NAK */
            if (tx_cmd ==  active)
             {
                tx_cmd = inactive;
                get_ccr_rq =active;
                tx_time_cnt =200*60; /* 60 sec. Time Out */
                tx_state = wait_NAK;
             }
            break;

        case wait_NAK : /* Waiting for a NAK character to begin Tx */

    /**********************************************************/
    /****  TRANSMITTER WAITING FOR A NAK TO BEGIN        *****/
    /****  TRANSMISSION.                                 *****/
    /****                                                *****/
    /****             Checks For Time Out                *****/
    /****                     or  NAK Received           *****/
    /**********************************************************/

            wn_status = check_wait ();      /* Time Out or NAK Rcvd? */
            switch (wn_status)  {

                case time_out :             /* If Time Out then Abort
                                               Transmission */
                    tx_state =tx_idle;
                    beep ();
                    prmsg ("Time OUT !!!! receiver not ready");
                    cll (tx_r, tx_c);
                    open_wind (tx_r,tx_c,"NONE");
                break;

                case waiting :              /* if no Time Out and no NAK
                                               rcvd then do nothing */
                break;

                case rx_NAK :
                                            /* If NAK received then Open
                                               file and advance to
                                               Transmit Packet State */

                    fp =fopen (tx_file_name,"rb" );
                    if (fp== NULL)
                    {
                     beep ();
                     prmsg ("ERROR !!! file does not exist");
                     cll (tx_r,tx_c);
                     open_wind (tx_r,tx_c, "none");
                     tx_state =tx_idle;
                    }
                 else
                        {
                        tx_state = tx_rdy;
                        txrflg = mkpkt;     /* First task for Tx
                                               is to Prepare Packet */
                        wn_status = 0;      /* Reset Wait_NAK Flag */
                        }
                break;
                }
        break;/* end  wait nak */
```

231928-42

82510 XMODEM Implementation (Continued)

```
PAGE  6        MAIN PROGRAM    ftp.c   82510 XMODEM

           case     tx_rdy  :
           /*********************************************************/
           /****  TANSMITTER READY TO TRANSMIT                 *****/
           /****                three stages of transmission   *****/
           /****                      prepare packet           *****/
           /****                      Int. Handler Transmitting *****/
           /****                      or retransmit request    *****/
           /*********************************************************/

                /* Any Control Character To Transmit? */
                if ((send_ccr_req == active) && (tx_req==0))
                   tx_req =ctl_chr;

                /* Which Stage of transmission ?*/
                switch (txrflg)
                {
                case mkpkt:                          /* Prepare Packet  */
                     if  (tx_req==0)
                     {
                      asmbpkt (pkts_sent,fp);        /* Assemble Packet */
                      cpy2buf ();
                      tx_req =pkt;                   /* Request Int. Handler
                                                        to Tx data in buffer */
                      txrflg =txmtg;                 /* Start Transmission */
                      tx_indx =0;
                      tx_i_enb ();                   /* Enable TxM and Tx FIFO
                                                        Interrupts */


                     }
                break;
                case txmtg  :
                     if (tx_req == 0)                /* Interrupt Handler Resets
                                                        this flag to 0, when 132
                                                        bytes are transmitted */
                     {
                       tx_indx =0;
                       prmsg ("packet transmitted");
                       get_ccr_rq =active;           /* Wait for ACK or NAK */
                       tx_time_cnt = 200*10;         /* 10 sec Time Out */
                       tx_state = wait_CC;           /* Wait for ctl Character */
                       txrflg =mkpkt;
                       tx_i_dis ();                  /* Disable TxM and Tx FIFO
                                                        Interrupts */
                     }
                     else                            /* Tx_req not reset then
                         prmsg ("transmitting");        still transmitting */
                break;
                case retx  :                         /* The Retransmit request is
                                                        issued by the Wait _CC
                                                        state */
                      outp((bpa+6),txen);            /* enable txm, flush tx fifo
                                                        & txm **/
                      tx_req = pkt;                  /* transmit Packet,pkt. in
                                                        buffer */
                      txrflg =txmtg;                 /* next task - ReTransmit */
                      tx_i_enb ();                   /* Enable TxM and Tx FIFO
                                                        Interrupts */
                break;
                }
          break; /* End tx rdy case */
```

231928–43

82510 XMODEM Implementation (Continued)

```
PAGE  7        MAIN PROGRAM    ftp.c  82510 XMODEM

        case wait_CC :
/**************************************************************/
/****                                                   *****/
/****  Transmitter State - Waiting For Ctl Char.        *****/
/****                                                   *****/
/****           NAK - requests retransmission           *****/
/****           ACK - Transmit Next Packet              *****/
/****                                                   *****/
/****                                                   *****/
/**************************************************************/

             wcst = check_wait ();                /* Check for one of the
                                                     Following events:
                                                      Time Out
                                                      NAK Received
                                                      ACK Received
                                                     or Still Waiting */
             switch (wcst)
              {
               case time_out :

                                                  /* If Time Out, then restart
                                                     Tx Timer. Abort if Time
                                                     Out count is greater than
                                                     ten */
                 if (tocnt >10)
                     {
                     wcst =0;
                     abort_tx ();
                     prmsg ("receiver not responding");
                     }
                 else
                     {
                      ++tocnt;                    /* Inc. Time Out Count */
                      tx_time_cnt =200*10;
                     }

               break;

               case waiting :                     /* if waiting, do nothing */
               break;

               case rx_NAK :                      /* If NAK or Corrupted
                                                     ctl-char. received */

               case rx_gen :
                 prmsg ("NAK received");
                 if (retx_cnt >10)                /* more than 10 attempts,
                                                     then Abort*/
                  {
                   retx_cnt =0;
                   tocnt =0;
                   abort_tx ();
                   prmsg ("Bad link transmission aborted");
                  }
```

231928-44

82510 XMODEM Implementation (Continued)

```
PAGE  8        MAIN PROGRAM    ftp.c   82510 XMODEM

                  else                             /* If Retransmit Count Not
                                                      exceeded then go back to
                                                      Transmit stage - task is
                                                      retransmit */
                   {
                    txrflg =retx;
                    ++ retx_cnt ;
                    tx_state =tx_rdy;
                   }
               break;

             case rx_ACK:                          /* ACK Received*/
                  prmsg ("ACK received");
                  retx_cnt=0;
                  tocnt = 0;
                  ++pkts_sent;
                  printf ("pkts_sent = %3u", pkts_sent);
                  if (eof ==false)                 /* If more data to transmit
                                                      then retrun to mkpkt
                                                      stage and tx new pkt. */
                     {
                      txrflg =mkpkt;
                      tx_state =tx_rdy;
                     }
                  else
                   {
                    prmsg ("sending EOT");         /* if end of file , then
                                                      send EOT */
                    ccr_to_tx = EOT;
                    tx_req =ctl_chr;
                    tx_i_enb ();
                    while (tx_req != 0);           /* wait for Int. Handler
                                                      to reset flag */
                    tx_i_dis ();
                    get_ccr_rq =active;            /* wait for Ack */
                    while (get_ccr_rq ==active);
                    prmsg ("EOT acknowledgement received");
                    if (ccr_to_get == ACK)         /* ACK rxd , Close File */
                    {
                       s = fclose (fp);
                       abort_tx();
                       prmsg ("file transmission complete");
                    }
                    tx_state =tx_idle;             /* Return to Idle */
                   }
             break;
             } /* /end wait_cc case */
         break;
         } /* end switch tx state */
```

231928-45

82510 XMODEM Implementation (Continued)

```
PAGE  9        MAIN PROGRAM    ftp.c   82510 XMODEM

/***************************************************************/
/*****           Process Rx STATE MACHINE            *****/
/*****           revision 0                          *****/
/*****                                               *****/
/***************************************************************/
        switch (rx_state)
        {
            case rx_idle:
            /***************************************************************/
            /****                                               *****/
            /**** RECEIVER IDLE:                                *****/
            /****                                               *****/
            /****                waits for user command         *****/
            /****                before sending NAKs            *****/
            /****                                               *****/
            /****                                               *****/
            /***************************************************************/

                if (rx_cmd == active)        /* If receive Command is issued
                                                then start Rx timer and change
                                                 Receiver state to ready */
                    {
                        rx_state = rx_rdy;
                        rx_time_cnt =200*10;
                        rx_cmd = inactive;
                    }
            break;

            case rx_rdy:
            /***************************************************************/
            /****                                               *****/
            /**** RECEIVER READY:                               *****/
            /****                 sends NAK upon Time Out       *****/
            /****                 or checks for SOH             *****/
            /****                 or EOT ctl-char.              *****/
            /****                                               *****/
            /****                                               *****/
            /***************************************************************/

                rxrflg =wait_rx ();      /* Checks Rx Timer and returns -
                                          Time Out if expired
                                          waiting if not expired
                                          SOH if SOH ccr received
                                          EOT if EOT ccr received */
                switch (rxrflg)
                {
                 case waiting :          /* If waiting then do nothing */
                 break;

                 case  SOH :             /* If SOH received, then go into
                                            data reception mode and change Rx
                                            Timer count to 4 secs */
                    ++ sohcnt;
                    rx_state =rx_pkt;
                    rx_time_cnt =200*4;  /* four second time out */
                    rxtocnt =0;
                break;

```

231928-46

82510 XMODEM Implementation (Continued)

```
PAGE 10        MAIN PROGRAM    ftp.c  82510 XMODEM

            case time_out:          /* if time out & not in the midst of
                                       packet reception then send NAK */
                if (( exp_pkt_num ==1) && (rx_byte_cnt ==0))
                {
                    prmsg ("rx time out !!!!! sending NAK");
                    if (send_ccr_req ==inactive)
                    {
                        ccr_to_tx =NAK;
                        send_ccr_req =active;
                    }
                }
                rx_time_cnt =200*10;
            break;

            case EOT:                                   /* If End Of Text rcvd,
                                                           and data rcvd then
                                                           send ACK and save all
                                                           packets received in
                                                           file */
                ++ eot_cnt;
                open_wind (22,50,"End of Text");
                if (exp_pkt_num >1)
                {
                  if (send_ccr_req == inactive) /* Send ACK */
                  { send_ccr_req =active;
                    ccr_to_tx =ACK;
                  }
                  rx_state =rx_idle;                /* Receiver Returns to
                                                       Idle */
                                                    /* create file */
                  rxfp =fopen (rx_file_name,"ab+");
                  rwst =fwrite (&rx_f_buf[0],128,exp_pkt_num-1,rxfp);
                  if (rwst <1)
                      {
                       prmsg ("Write file error ");
                       printf ("error = %4u",(rwst=ferror(rxfp)));
                      }
                  rval =fclose (rxfp);
                  if (rval ==0)
                      prmsg ("file received");
                  else
                    prmsg ("Error in closing file ");

                }
            break;
            }
        break;

    /*.PA*/
        case rx_pkt: /* Packet reception */
        /************************************************************/
        /****                                                *****/
        /****   RECEIVE PACKET STATE                         *****/
        /****                                                *****/
        /****               checks for Time Out              *****/
        /****               or 131 bytes received            *****/
        /****               which signals the end of packet  *****/
        /****                                                *****/
        /************************************************************/
```

231928–47

82510 XMODEM Implementation (Continued)

```
PAGE 11        MAIN PROGRAM    ftp.c   82510 XMODEM

          /* If valid Rx Time Out , i.e. no data received for 4 secs then
             check Rx FIFO for characters and read if any available */
          if ((rx_time_cnt ==0) && (rx_byte_cnt <131))
          {
             rxfl = ((inp (bpa +4) & 0x70)/ 0x10);          /* check Rx FIFO
                                                              Level */
             if ((rxtocnt >= 10) && (rxfl <=0))             /* if more than
                                                              10 attempts
                                                              and no data
                                                              then abort
                                                              transmit */
             {
                 rx_state =rx_idle;
                 prmsg (" Receiver Time Out, no DATA");
                 rxtocnt =0;
             }

             else
            /* otherwise restart Rx Timer, and read data from 510 */
            {
             if (rxfl != 0)                         /* Rx FIFO level > 0 */
             {
                 rx_time_cnt =200*5;
                 rxfl = ((inp ( bpa +4) & 0x70)/0x10);
                 while ( rxfl != 0)                 /* Read from FIFO */
                 {
                     rxdata [rx_byte_cnt] = inp (bpa);
                     ++ rx_byte_cnt;
                     ++ rxfcnt;
                     -- rxfl;
                 }
                 rxtocnt = 0;
             }
             else
             {
                 ++ rxtocnt;                        /* inc. receive TimeOut
                 rx_time_cnt =200*4;                   Count */
             }
            }
          }
          else
          {
             if (rx_byte_cnt == 131)                /* Packet Received */
             {
                 rx_byte_cnt =0;
                 rxtocnt =0;
                 pkst =chkpkt (exp_pkt_num);        /* Check Packet */
                                                    /* returns EOK if Packet
                                                       without errors */
      /*.PA*/
                 if ((pkst == eok) || (pkst ==eold))
                 {
                     prmsg ("sending ACK");
                     for (i=0; i<128; i++)
                         rxbuf [i] = rxdata [i+2];
                     /** write packet to buffer  **/
                     if (pkst ==eok)
                     {
                                                    /* copy to main file
                                                       buffer */
```

231928-48

82510 XMODEM Implementation (Continued)

```
PAGE 12        MAIN PROGRAM    ftp.c  82510 XMODEM

                            buf_cpy (exp_pkt_num);
                            ++ exp_pkt_num;
                        }
                        else
                            prmsg ("old packet retransmitted");
                    }

                    else
                        sh_pkt_param ();            /* If error then show
                                                       packet #, chksum and
                                                       packet complement */
                    rx_state = rx_rdy;
                    mskint4 ();                     /* Enable Ctl-Chr int*/
                    set_bank (00);
                    outp ((bpa+1),(inp(bpa+1)|ccien));
                    set_bank (01);
                    enbint4 ();
                    send_ccr_req =active;           /* Send ACK */
                    ccr_to_tx =ACK;

                }

            }
            break;

        } /** end switch rx state **/


    } /* end else */

} /* end while quit */

rst510 ();                                  /* reset 82510 */
outp ((bpa + 1),00);                        /* disable 82510 interrupts */


cmd = 0x10;                                 /* disable 8259A interrupt */
v=inp (0x21);                               /*       00010000          */
cmd = (v | cmd);
outp  (0x21,cmd);
clr ();
ecode  = 0;
_exit (ecode);

}/* end main */

```

231928-49

82510 XMODEM Implementation (Continued)

```
PAGE 13        MAIN PROGRAM   ftp.c  82510 XMODEM


rst510 ()
/*************************************************************/
/****                                                   *****/
/****     RESET 82510 to default wake up mode           *****/
/****                                                   *****/
/****                                                   *****/
/****                                                   *****/
/*************************************************************/

{
set_bank (01);
outp ((bpa+7),0x10);
}


menu ()
/****************************************************/
/**      displays the menu on the                  **/
/**      screen.                                   **/
/**                                                **/
/**                                                **/
/**                                                **/
/****************************************************/
{


      open_wind (1,1,"baud rate");
      printf (" = 1200  ");
      open_wind (1,22,"char. size");
      printf (" = 8 bits");
      open_wind (1,45 , "Parity");
      printf (" disabled");
      open_wind (1,68, "Stop Bits");
      printf (" = 2");
      mv_curs (2,1);
      printf ("user messages :");
      mv_curs (10,15);
      printf ("(1) TRANSMIT FILE : ");
      OPEN_WIND (tx_r,tx_c,"none");
      mv_curs (12,15);
      printf ("(2) RECEIVE FILE : ");
      OPEN_WIND (rx_r,rx_c,"none");
}

```

231928-50

82510 XMODEM Implementation (Continued)

```
PAGE 14        MAIN PROGRAM    ftp.c   82510 XMODEM


init      ()
/****************************************************/
/**   Intializes Software and Configures          **/
/**   the 82510. Also sets up the interrupt       **/
/**    Handler.                                    **/
/**                                                **/
/**                                                **/
/****************************************************/

{
tx_time_cnt =200;
rx_time_cnt =2000;
initpack ();
clms ();
init_ih ();                                  /* Set up interrupt handler */
config_510 ();                               /* Configure 82510 */
set_bank (01);                               /* Switch to Bank one for operation */

}
initpack ()
/****************************************************/
/**                                                **/
/**       Intializes Tx Buffer to NULs             **/
/**                                                **/
/****************************************************/

{
int       i;

txpack.head = SOH;
rxpack.head =SOH;
txpack.pack_num =0;
rxpack.pack_num =0;
txpack.pack_cmpl = 0;
rxpack.pack_cmpl = 0;
for (i=0; i <129; i++)
     {
           rxpack.buffer[i] =NUL;
           txpack.buffer[i] =NUL;
     }
txpack.chksm =0;
rxpack.chksm =0;
}

enbint4 ()
/****************************************************/
/**                                                **/
/**       Enables INT4  in the 8259A               **/
/**                                                **/
/****************************************************/

{
int       int_enb = 0xEF;
int       v;

v=inp (ip01);                      /* 11101111 */
int_enb = (v & int_enb);
outp  (ip01,int_enb);
}
```

231928–51

82510 XMODEM Implementation (Continued)

```
PAGE 15      MAIN PROGRAM    ftp.c   82510 XMODEM

mskint4 ()
/****************************************************/
/**                                               **/
/**     Masks INT4 in the 8259A                   **/
/**                                               **/
/****************************************************/
{
int      int_dis = 0xEF;
int      v;

v=inp (ip01);                        /* 00010000 */
int_dis = (v | 0x10);
outp  (ip01,int_dis);
}

config_510 ()
/****************************************************/
/**                                               **/
/**     Configure the 82510                       **/
/**                                               **/
/****************************************************/
{
int val;

set_bank (02);                  /**********************************************/
val = 0x00;                     /* IMD - Rx FIFO depth =4, auto ack,normal   */
outp ((bpa + 4), val);          /* local loopback                            */
val =0x78    ;                  /* RMD - ASCII CCR,disable dpll,7/16 sampl   */
outp ((bpa +7),val);            /* window,absolute start bit sampling        */
val =0x00         ;             /* TMD - manual  mode, 2 stop bits           */
outp ((bpa +3),val);            /* no 9-bit char, no s/w parity              */
val =0x30;                      /* FMD - Rx fifo Threshold = 3               */
outp ((bpa+1),val);             /* Tx fifo threshold =0                      */
val =0x80;                      /* RIE - Enable rx interrupts                */
outp ((bpa+6),val);             /*                                           */
set_bank (03);                  /* MODEM CONFIGURATION                       */
val = 0x50;                     /* CLCF - 16X, BRGA                          */
outp ((bpa),val);               /*                                           */
val =0xd8;                      /* BBL   - for 5ms base                      */
set_dlab (03);                  /*                                           */
outp ((bpa), val);              /*                                           */
val =0xb4;                      /*                                           */
outp ((bpa+1),val);             /* BBH  - for 5 ms base                      */
reset_dlab (03);                /*                                           */
val = 0x00;                     /* BBCF - sys clk source, timer mode         */
outp ((bpa+3),val);             /*                                           */
val =0x02;                      /* TMIE - Timer B interrupt enable           */
outp ((bpa+6),val);             /*                                           */
set_bank (00);                  /* BANK 0 FOR GENERAL CONFIG                 */
val =blkenb;                    /* GER -  enable timer, rx, CCR              */
outp ((bpa+1),val);             /* block interrupts                          */
val = 0x07;                     /* LCR - disable parity, 8 bit char          */
outp ((bpa+3),val);             /*                                           */
set_dlab (00);                  /*                                           */
val = 0xE0;                     /* BRGA divisor =01E0H for 1200              */
outp (bpa,val);                 /**********************************************/
val = 0x01;
outp ((bpa+1),val);
reset_dlab (00);
}
```

231928-52

82510 XMODEM Implementation (Continued)

```
PAGE 16        MAIN PROGRAM    ftp.c   82510 XMODEM

set_dlab (bank)
/***************************************************/
/**                                               **/
/**      Set DLAB bit to allow access to          **/
/**      Divisor Registers                        **/
/**                                               **/
/***************************************************/

int bank;
{
int      inval;
set_bank (00);
inval = inp(bpa +3);
inval =inval | 0x80;                /* set dlab in LCR*/
outp ((bpa+3),inval);
set_bank (bank);
}

reset_dlab (bank)
/***************************************************/
/**                                               **/
/**      Reset DLAB bit of LCR                    **/
/**                                               **/
/***************************************************/

int bank;
{
int      inval;
set_bank (00);
inval = inp(bpa +3);
inval = (inval & 0x7f);             /* dlab = 0 in LCR*/
outp ((bpa+3),inval);
set_bank (bank);
}
```

231928-53

82510 XMODEM Implementation (Continued)

```
PAGE 17      MAIN PROGRAM    ftp.c   82510 XMODEM

/*****************************************************************/
/****                                                      *****/
/****     82510 interrupt service routine                  *****/
/****                                                      *****/
/****     82510 Interrupt sources:                         *****/
/****                           TxM     TX FIFO            *****/
/****                           CCR     RX FIFO            *****/
/****                                   TIMER B            *****/
/****                                                      *****/
/****    Identifies and services the 82510 interrupt       *****/
/****    source requesting service.                        *****/
/****                                                      *****/
/****                                                      *****/
/*****************************************************************/

isr_510 ()
{
int        source ;
int        cmd_b;
int        st_b;
int        i;
int        ctlc;
int        flgs;
int        girval;                          /* Stores Temp. Value of GIR */
int        rxflvl;
int        tx_char;

girval =inp (bpa+2);                        /* Save Bank register in temp.
                                               location */
outp ((bpa+2),0x20)
source = getsrc ();                         /* Get Vector From GIR 123 */
intvec =source;
switch (source)  {                          /* Service the Source */

      case timer :
/***************************************************************/
/****                                                    *****/
/****    TIMER SERVICE ROUTINE                           *****/
/****                    decrements tx counter           *****/
/****                    decrements rx counter           *****/
/****                                                    *****/
/***************************************************************/

          st_b = inp (bpa+3);                /* Decrement Transmit Counter */
          if (tx_time_cnt >0)
              tx_time_cnt =tx_time_cnt - 1;
          if (rx_time_cnt >0)                /* Decrement Receive Counter */
              rx_time_cnt =rx_time_cnt - 1;
          cmd_b = 0x22;
          outp ( (bpa+3),cmd_b );            /* restart timer */
          outp ((bpa+7),0x08);               /* manual ack */
          break;
```

231928-54

82510 XMODEM Implementation (Continued)

```
PAGE 18        MAIN PROGRAM    ftp.c   82510 XMODEM

        case txm   :
        case txf   :
        /***********************************************************/
        /****                                                 *****/
        /****  TRANSMITTER SERVICE ROUTINE                    *****/
        /****                                                 *****/
        /****                transmits Four characters        *****/
        /****                and resets tx_req flag when      *****/
        /****                whole packet transmitted         *****/
        /****                                                 *****/
        /***********************************************************/

                if (tx_req >0)                     /* If data to send */
                {
                    if (tx_req == pkt)             /* request to send Packet */
                    {
                        for (i =0 ; i<4 ;i++)
                        {
                            tx_char = txdata [i + tx_indx];
                            outp (bpa,tx_char);
                        }

                        tx_indx +=4;
                        tx_byte_cnt +=4;

                        if (tx_indx )=132)         /* if 132 char. sent then */
                            tx_req = 0;            /* reset Tx request */

                    }
                    else
                    {
                        if (tx_req ==ctl_chr)      /* if ctl char. transmission
                                                      requested , then transmit the
                                                      char. in ccr_to_tx */
                             outp (bpa, ccr_to_tx);
                        tx_req =0;
                    }
                }
                else
                    {                              /* if no data to transmit */
                                                   /* then disable tx interrupts */
                        set_bank (00);
                        outp ((bpa+1), (inp(bpa) &txidb));
                        set_bank (01);
                    }
            outp ((bpa+7),0x08);                   /* issue manual acknowledge */
        break;
```

231928-55

82510 XMODEM Implementation (Continued)

PAGE 19 MAIN PROGRAM ftp.c 82510 XMODEM

```
case ccr  :
  /*****************************************************************/
  /****                                                      *****/
  /****    Control Character Service Routine                *****/
  /****                                                      *****/
  /****                  if control char = NAK or ACK         *****/
  /****                      inform    transmitter            *****/
  /****                  if SOH  or EOT                       *****/
  /****                      inform receiver                  *****/
  /*****************************************************************/

             ++ccrcnt;
             flgs =inp (bpa +5);         /* read RST register to service
                                            RxM interrupt */
             flgs =inp (bpa+1);
             ctlc =inp (bpa);
             if ((flgs & 0xFF) ==0x48)  /* if no errors and ctl. char */
             {                          /* then process control char. */
                                        /* and send to tx or rx state */
             switch (ctlc)
              {
                 case NAK:
                 case ACK:
                     if (get_ccr_rq == active)
                         {               /* inform transmitter that
                                            ctl. char. received */
                             get_ccr_rq =inactive;
                             ccr_to_get =ctlc;
                         }

                  break;

                  case SOH:
                  case EOT:
                      if (ctlc ==SOH)     /* if SOH disable CCR int. */
                      {
                        set_bank (00);
                        outp ((bpa+1),(inp(bpa+1)& ccidb));
                        set_bank (01);
                      }
                      if (rx_state == rx_rdy)  /* if receiver waiting for
                                                  SOH and ready to rcv
                                                  then inform receiver of
                                                  a valid ctl. char. */
                       {
                          ctl_rxd_flg =active;
                          rx_ctl_chr =ctlc;
                       }

                  break;

              }
             }
             outp ((bpa+7),0x08);             /* issue manual ack. */
  break;
```

231928-56

82510 XMODEM Implementation (Continued)

```
PAGE 20        MAIN PROGRAM    ftp.c   82510 XMODEM


        case rxf    :
        /***************************************************************/
        /****                                                     *****/
        /****  Rx FIFO SERVICE ROUTINE                            *****/
        /****                                                     *****/
        /****            Reads four bytes                         *****/
        /****            Byte Count indicates packet rcvd.        *****/
        /****                                                     *****/
        /****                                                     *****/
        /***************************************************************/


            /* RXF not checked for errors, since checksum is already used*/

                    rx_time_cnt =200*5;      /* reset Rx Timer  to indicate
                                                char. received before time out */

                    rxflvl = ((inp ( bpa +4) & 0x70)/0x10);
                    while ( rxflvl != 0)     /* Check Rx FIFO level and read
                                                data if FIFO not empty */
                    {
                        rxdata [rx_byte_cnt] = inp (bpa);
                        ++ rx_byte_cnt;
                        ++ rxfcnt;
                        -- rxflvl;
                    }
                    outp ((bpa+7),0x08);     /* issue manual acknowledge */
        break;
        default    :
                                             /* if invalid source then issue a
                                                manual acknowledge */
                    outp ((bpa+7),0x08);
        break;
        }
    outp ((bpa+1),girval);                   /* Restore Original value of Bank
                                                register to return the 82510 to
                                                original Bank  */

    outp (ip00,eoi);                         /* issue end of int. to 8259*/
    }


    Set_bank (bank_num)
    int       bank_num;
    ****************************************************************/
    /****   PROCEDURE SET_BANK                                *****/
    /****   switches 82510 register bank to                   *****/
    /****   given value.                                      *****/
    /**************************************************************/
    {
    int       port;
    int       bank_reg_val;

    bank_reg_val     =bank_num * 0x20;
    port = gir_addr +bpa;
    outp (port, bank_reg_val);            /* output value to bank register */
    }

```

231928-57

82510 XMODEM Implementation (Continued)

```
PAGE 21        MAIN PROGRAM   ftp.c  82510 XMODEM

getsrc ()
/****************************************************/
/**      read GIR and returns the                  **/
/**      source Vector                             **/
/**                                                **/
/**      Timer      - 05 Hex                       **/
/**      Tx Machine - 04 Hex                       **/
/**      CCR        - 03 Hex                       **/
/**      Rx FIFO    - 02 Hex                       **/
/**      Tx FIFO    - 01 Hex                       **/
/**                                                **/
/****************************************************/

{
int      v,src;

v=inp (bpa +2);                        /* read GIR */
src = v & 0x0E;                        /* Mask out all bits except for
                                          bits 1,2 and 3 */
src = src/2;
return(src);
}

process_cmd ()
/************************************************************/
/****                                                  *****/
/**** PROCESS COMMAND                                  *****/
/****            Processes User commands               *****/
/****            1 - Transmit                          *****/
/****            2 - Receive                           *****/
/****            * - Reset 82510                       *****/
/****            0 - quit                              *****/
/****            r - Reinitialize 82510                *****/
/****            ! - system monitor                    *****/
/****                                                  *****/
/****                                                  *****/
/************************************************************/


{
int      r;
int      exflg =false ;
int      excp ;

     r = getch ();
     switch (r)  {

          case '0' :
               exflg = true;                /* exit */
               break ;
```

231928-58

82510 XMODEM Implementation (Continued)

```
PAGE 22        MAIN PROGRAM    ftp.c   82510 XMODEM

                case '1' :
                    if (tx_state == tx_idle)         /* Transmit Command only
                                                        accepted if idle */
                    {
                        CLMS ();
                        CLL (tx_r,tx_c);
                        MV_CURS (tx_r,tx_c);
                        printf ("file :");           /* Get name of file to Tx */
                        scanf ("%s", &tx_file_name);
                        cll (tx_r,tx_c);
                        open_wind (tx_r,tx_c,"transmitting");
                        open_wind (tx_r,tx_c+14,tx_file_name);
                        tx_cmd = active;             /* Activates flag to signal
                                                        Transmit idle state */
                    }
                    else
                    {
                     beep ();
                     prmsg ("transmission in progress");
                    }
                break;
                case '2' :
                    CLMS ();
                    CLL (rx_r,rx_c);
                    MV_CURS (rx_r,rx_c);
                    printf ("file :");               /* Get rx file name */
                    scanf ("%s", &rx_file_name);
                    cll (rx_r,rx_c);
                    open_wind (rx_r,rx_c,"enabled");
                    open_wind (rx_r,rx_c+14,rx_file_name);
                    rx_cmd =active;                  /* Activate flag to signal
                                                        rx state machine */
                    break;
                case '*' :
                    rst510 ();                       /* reset 82510 */
                    open_wind (24,30,"device reset");
                    break;
                case 'r' :
                    rst510 ();
                    init ();                         /* reinitialize 82510 */
                    enbint4 ();
                    beep ();
                    prmsg (" 82510 reinitialized");
                    break;
                case '!' :
                     excp = system ("d:\micom");
                default:
                     BEEP ();
                     prmsg ("incorrect command, reenter");
                     break;
            }
            if (exflg == true)                       /* if exit command issued,
                                                        then quit program */
                return (true);
            else  return (false);
    }  /* end of command processing */
```

231928-59

82510 XMODEM Implementation (Continued)

```
PAGE 23        MAIN PROGRAM    ftp.c   82510 XMODEM

asmbpkt (pkts_sent,fp)
/****************************************************************/
/**      Reads file to be transmitted and puts                **/
/**      the data into the proper xmodem packet format        **/
/****************************************************************/

int      pkts_sent;                         /* this value is used to
                                               get the next pkt # */
FILE     *fp;
{
int      sum =0;
int      i,blkcnt;
int      st,ft;
char     cpkt,cpkcmp;

 blkcnt =fread (&txpack.buffer[0],128,1, fp);    /* read 128 bytes */
 if (blkcnt <1)
   {
     if ((st=feof(fp)) >0 && !(ft=ferror(fp)))
      {
        eof = true;                         /* if end of file then
                                               signal EOF */
        beep ();
        prmsg ("EOF !!!!!!!!!! ");
      }
         else
        if (ft >0)
           {
             beep ();
             prmsg ("READ ERROR !!!!!!!!!!!");
             tx_state=tx_idle;
           }
   }

 cpkt =pkts_sent +1;
 txpack.pack_num = cpkt;
 cpkcmp =~txpack.pack_num;
 txpack.pack_cmpl = cpkcmp;                 /* one's complement of
                                               packet number */
 for (i=0; i <128; i++)
     sum = sum+txpack.buffer[i];
 txpack.chksm = sum % 255;                  /* checksum calculated */

}


cpy2buf ()
/********************************/
/** copy packet to tx buffer **/
/********************************/
{
int i;

txdata [0] =txpack.head;
txdata [1] =txpack.pack_num;
txdata[2] =txpack.pack_cmpl;
for (i=0; i <128; i++)
    txdata [i+3] = txpack.buffer [i];
txdata [131] =txpack.chksm;
}
```

231928-60

82510 XMODEM Implementation (Continued)

```
PAGE 24        MAIN PROGRAM    ftp.c   82510 XMODEM


check_wait ()
/*************************************************************/
/**** PROCEUDRE CHECK_WAIT                              *****/
/****                                                   *****/
/****             checks  Tx Timer, ccr_to_get and      *****/
/****                     get_ccr_req  and returns:     *****/
/****                                                   *****/
/****              Time Out - Tx Timer = 0              *****/
/****              rx_ACK   - Ack received              *****/
/****              rx_NAK   - Nak received              *****/
/****              waiting  - tx Timer not expired      *****/
/****                                                   *****/
/****                                                   *****/
/*************************************************************/
{

 if ((!tx_time_cnt) && (get_ccr_rq ==active))        /* if tx Timer expired
                                                        and still waiting
                                                        for control char,then
                                                        Time out */
          return (time_out);
 else
     if (get_ccr_rq == inactive)                    /* Ctl-Char rcvd then
                                                       return status */
         {
          switch (ccr_to_get)
          {
           case ACK :
                  return (rx_ACK);
           break;

           case NAK :
                  return (rx_NAK);
           break ;

       default:
                 return (rx_gen);                    /* corrupted ctl char */
           break;
          }
         }
     else
          if ((tx_time_cnt >0) && (get_ccr_rq ==active))
              return (waiting);
}


```

231928–61

82510 XMODEM Implementation (Continued)

```
PAGE 25        MAIN PROGRAM    ftp.c   82510 XMODEM

abort_tx ()
/***************************************************************/
/****                                                    *****/
/****     Abort transmission, reintialize                *****/
/****                          Transmitter               *****/
/****                          Flags                     *****/
/****                                                    *****/
/***************************************************************/

{
            eof =false;
            txrflg = mkpkt;
            quit =false;
            key = 0;
            tx_state =tx_idle;
            tx_cmd = inactive;
            send_ccr_req = inactive;
            tx_indx = 0;
            tx_req =inactive;
            ccr_to_tx = 0;
            tx_byte_cnt =0;
            pkts_sent =0;
            tx_time_cnt =0;
            get_ccr_rq =0;
            ccr_to_get  =0;
            set_bank (00);
            outp ((bpa+1),0x27);
            set_bank (001);
            outp((bpa+6),0x0D);
            tx_state =tx_idle;
            prmsg ("transmitter reset");
}


wait_rx ()
/***************************************************************/
/****                                                    *****/
/****    WAIT_RX:                                        *****/
/****            checks rx timer, and returns the        *****/
/****            following value :                       *****/
/****                 SOH - SOH received                 *****/
/****                 EOT - EOT received                 *****/
/****                 time out - rx timer expired        *****/
/****                 waiting - waiting for event        *****/
/****                                                    *****/
/***************************************************************/

{
      if ((ctl_rxd_flg == active) && (rx_time_cnt !=0))
      {
            ctl_rxd_flg =inactive;
            return ( rx_ctl_chr);
      }
      else
            if ( rx_time_cnt == 0)
                return (time_out);
            else
                return (waiting);
}
```

231928-62

82510 XMODEM Implementation (Continued)

```
PAGE 26      MAIN PROGRAM    ftp.c   82510 XMODEM

chkpkt (pknum)
/***********************************************/
/**     verifies the checksum and packet     **/
/**     number of the received packet        **/
/**     returns a status code                **/
/**                                          **/
/**     EOK      - Packet Ok                 **/
/**     EPKNUM   - Error in packet number    **/
/**     ECHKSUM  - Error in Check Sum        **/
/**     EPKCMPL  - Erro in packet complement **/
/**                                          **/
/***********************************************/
int pknum;
{
int i;
int ckm;
int sum = 0;
char    cmpl,rxcmpl,cpk,chrckm;


      cpk =pknum;
      if (cpk ==rxdata[0])                        /* packet number correct */
       {
        cmpl = rxdata [0];
        rxcmpl =rxdata [1];
        if (rxcmpl == ~cmpl )                     /* packet number complmt */
          {
             for (i=2; i<130; i++)
                 sum = sum +rxdata [i];
             ckm = sum % 255;
             chrckm = ckm;
             pk_chksm =chrckm;
             if (chrckm == rxdata [130])          /* checksum correct */
                 return (eok);
             else
                 return (echksm);

           }
           else
             return (epkcmp);
       }
           else
          {
           if ((rxdata [0] == cpk -1) && (cpk >1))     /* old packet number
                                                          received */
             return (eold);
           else
             return (epknum);
           }
 }
```

231928-63

82510 XMODEM Implementation (Continued)

PAGE 27 MAIN PROGRAM ftp.c 82510 XMODEM

```
/*****************************************************************/
/****          tx_i_dis      PROCEDURE                      *****/
/****    Disables txm and Tx FIFO interrupts                *****/
/****    from GER register                                  *****/
/*****************************************************************/
tx_i_dis ()
{
mskint4 ();                                      /* disable interrupts */
set_bank (00);                                   /* switch to bank zero */
outp ((bpa+1),(inp(bpa+1) & txidb));             /* mask out interrupts in
                                                    GER - TxM and Tx FIFO */
set_bank (01);                                   /* set bank one */
enbint4 ();                                      /* enable interrupts */
}/*****************************************************************/
/****               tx_i_enb     PROCEDURE                  *****/
/****    enables the TXM and TX FIFO Interrupts             *****/
/****    from GER register                                  *****/
/*****************************************************************/
tx_i_enb ()
{
mskint4 ();                                      /* disable interrupts */
set_bank (00);                                   /* switch to bank zero */
outp ((bpa+1),(inp(bpa+1) | txien));             /* enable TXM AND TX FIFO */
set_bank (01);                                   /* return to bank one */
enbint4 ();                                      /* enable interrupts */
}
sh_pkt_param ()
/****************************************************/
/** Displays the parametrs of the Received       **/
/** packet number, and the expected parameters **/
/****************************************************/
{
++ bad_pkt_cnt;
prmsg ("sending NAK ");
printf (" error = %5u",pkst);
mv_curs (13,1);
printf ("exptd pkt # = %3u",exp_pkt_num);
mv_curs (14,1);
printf ("rxd pkt # = %3u", rxdata[0]);
mv_curs (13,40);
printf ("expd pkt cmpl = %#X", (~rxdata[0]));
mv_curs (14,40);
printf ("rxd pkt complement = %#X", rxdata[1]);
mv_curs (15,1);
printf ("rxd chksum = %#X", rxdata[130]);
mv_curs (15,40);
printf ("expd chksum = %#X",pk_chksm);
send_ccr_req =active;
ccr_to_tx = NAK;
}
```

231928-64

82510 XMODEM Implementation (Continued)

```
PAGE 28        MAIN PROGRAM   ftp.c  82510 XMODEM

/****************************************/
/** PROCEDURE BUF_CPY                  **/
/**      copies packet to ram buffer   **/
/****************************************/
buf_cpy (packt_id)
int packt_id;
{
int      i;
int      indx =0;

indx = (packt_id-1) *128;
if (indx < (32000 - 129))   /* No overwrite of buffer */
{
     for (i=0; i<128; i++)
         rx_f_buf [indx+i] = rxbuf [i];
}
else
     prmsg ("file too big, cannot save in memory");
}
```

231928-65

82510 XMODEM Implementation (Continued)

```
PAGE  1         DEFINITION FILE    ftp.def   82510 XMODEM

#define s1  "82510 FTP x000 6/30/86"             /* sign on message */
#define bpa 0x3f8                                /* Base address */
#define gir_addr     02
#define esc1 27                                  /* escape char. in hex */
#define bel  07
#define msg_c 17                                 /* coordinates of the
                                                    message line */
#define msg_r 2
#define tx_c  35                                 /*
#define tx_r  10
#define rx_c  35
#define rx_r  12                                     coordinates
#define so_c  50
#define so_r  24                                 */
#define false  0
#define true   1
#define active 1
#define inactive 0
#define ctl_chr 2                                /* control char transmit */
#define pkt 1                                    /* send packet */
#define eok 5555                                 /* packet received ok */
#define echksm 5500                              /* checksum error */
#define epkcmp  5501                             /* packet compl incorrect */
#define eold    5502                             /* old pack num received */
#define epknum 5503                              /* invalid packet # rcvd. */

/***************************/
/*** tx state defintions***/
/***************************/

#define tx_idle      000
#define wait_NAK     001
#define TO_err_60    002
#define tx_rdy       003
#define tx_packet    004
#define wait_CC      005
#define tx_pk_comp   006
#define to_err       007
#define txen         0x02
#define mkpkt        111                         /* Transmit packet stages */
#define txmtg        112
#define retx         113
#define waiting      114

/***************************/
/**  rx state definition **/
/***************************/

#define rx_idle      000
#define rx_rdy       001
#define rx_pkt       002


/***************************/

#define      time_out     90                     /* rx state signal values */
#define      rx_NAK       91
#define      rx_ACK       92
#define      rx_gen       93

```

231928-66

82510 XMODEM Implementation (Continued)

```
PAGE  2        DEFINITION FILE    ftp.def   82510 XMODEM


/********************************/
/**      Protocol Control   ****/
/**      characters         ****/
/********************************/


#define NAK      0x14                    /* Negative Ack */
#define ACK      0x06                    /* Positive Ack */
#define SOH      0x01                    /* Start of Header */
#define EOT      0x04                    /* End of Text */
#define CAN      0x18
#define NUL      0x00

/********************************/
/**      interrupt source   ****/
/********************************/

#define timer    05                      /* 82510 int. vectors */
#define txm      04
#define ccr      03
#define rxf      02
#define txf      01
#define txien    0x12                    /* unmask TxM and Tx FIFO */
#define txidb    0x2D                    /* mask  TxM and Tx FIFO */
#define ccien    0x04                    /* enable CCR interrupts */
#define ccidb    0x33                    /* mask CCR int */
#define blkenb   0x25                    /* enable, block interrupts
                                            through GER for 82510 */

/*******************************/
/**      8259A values *********/
/*******************************/

#define eoi      0x20                    /* end of interrupt */
#define ip00     0x20                    /* 8259A port 0 */
#define ip01     0x21                    /* 8259A port 1 */
```

231928-67

82510 XMODEM Implementation (Continued)

```
PAGE  1        CRT I/O ROUTINES    cio.c    82510 XMODEM

#include "ftp.def"

CLR()
/*************************************************************/
/****                                                   *****/
/****     PROCEDURE CLR                                 *****/
/****                                                   *****/
/****                    clears screen                  *****/
/****                                                   *****/
/****                                                   *****/
/****                                                   *****/
/*************************************************************/

{
int escchr = esci;

    putch (escchr);
    printf ("[2J");

}


VOFF ()
/*************************************************************/
/****                                                   *****/
/****       PROCEDURE VOFF                              *****/
/****                                                   *****/
/****                  Turns Reverse Video OFF          *****/
/****                                                   *****/
/****                                                   *****/
/****                                                   *****/
/*************************************************************/

{
int escchr = esci;

    putch (escchr);
    printf ("[0m");

}


RVON ()
/*************************************************************/
/****                                                   *****/
/****    PROCEDURE RVON                                 *****/
/****                                                   *****/
/****                 Reverse Video ON                  *****/
/****                                                   *****/
/****                                                   *****/
/****                                                   *****/
/*************************************************************/

{
int escchr = esci;

    putch (escchr);
    printf ("[7m");
}

```

231928-68

82510 XMODEM Implementation (Continued)

```
PAGE  2        CRT I/O ROUTINES    cio.c    82510 XMODEM

OPEN_WIND (row,col,stg)
int      row;
int      col;
char     stg[];
/***************************************************************/
/****                                                     *****/
/****     PROCEDURE OPEN_WIND                             *****/
/****                                                     *****/
/****                  prints a string in reverse video   *****/
/****                  at the given location              *****/
/****                                                     *****/
/****                                                     *****/
/***************************************************************/

{

    MV_CURS (row, col);
    RVON ();
    printf ("%s",stg);
    VOFF();

}
BEEP ()
/***************************************************************/
/****                                                     *****/
/****   PROCEDURE BEEP                                    *****/
/****                                                     *****/
/****                  produces a beep                    *****/
/****                                                     *****/
/****                                                     *****/
/****                                                     *****/
/***************************************************************/

{
int belchr = bel;

    putch (belchr);

}

CLL(row,col)
int row;
int col;
/***************************************************************/
/****                                                     *****/
/****     PROCEDURE CLL                                   *****/
/****                                                     *****/
/****                  clear line at given coordinate     *****/
/****                                                     *****/
/****                                                     *****/
/****                                                     *****/
/***************************************************************/

{
int escchr = esci;
    MV_CURS (row, col);
    putch (escchr);
    printf ("[K");
}

```

231928-69

82510 XMODEM Implementation (Continued)

```
PAGE  3         CRT I/O ROUTINES     cio.c    82510 XMODEM


CLMS()
/**************************************************************/
/****                                                    *****/
/****       PROCEDURE CLMS                               *****/
/****                                                    *****/
/****                  clear message line                *****/
/****                                                    *****/
/****                                                    *****/
/****                                                    *****/
/**************************************************************/

{
    CLL (msg_r,msg_c);
}

prmsg (msg)
char    msg [];
/**************************************************************/
/****                                                    *****/
/****    PRINTS MESSAGE AT MESSAGE LINE                  *****/
/****                                                    *****/
/****                                                    *****/
/****                                                    *****/
/****                                                    *****/
/****                                                    *****/
/**************************************************************/

{
    clms ();
    printf (" %s", msg);
}

CLLC ()
{
int escchr = esci;
    putch (escchr);
    printf ("[K");
}

MV_CURS (x,y)
/**************************************************************/
/****                                                    *****/
/****  PROCEDURE MV_CURS                                 *****/
/****                                                    *****/
/****                 moves cursor to specified          *****/
/****                 location.                          *****/
/****                                                    *****/
/**************************************************************/

int x;
int y;
{
int escchr = esci;

    putch (escchr);
    cprintf ("[%u;%uH",x,y);
}
```

231928-70

82510 XMODEM Implementation (Continued)

```
PAGE  1     ASM 86 INTERRUPT INIT. ihl.asm 82510 XMODEM

NAME            ftpih

DGROUP  GROUP    DATA
DATA     SEGMENT WORD PUBLIC 'DATA'
      ASSUME  DS:DGROUP
DATA     ENDS

EXTRN           isr_510:far

_PROG    SEGMENT BYTE PUBLIC 'PROG'
      ASSUME  CS:_PROG

PUBLIC          init_ih
PUBLIC          ih510

init_ih         PROC     far
        push    BP
        push    DX
        push    AX
        push    DS
        mov     DX, OFFSET ih510
        push    CS
        pop     DS
        mov     AH,25H              ;DOS vector setup call
        mov     AL,0CH              ;COM1  vector
        INT     21H                 ;DOS system call
        pop     DS
        pop     AX
        pop     DX
        pop     BP
        ret
init_ih         ENDP

ih510           PROC     far
        push    BP
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI
        push    DI
        push    DS
        push    ES
        mov     AX, DGROUP
        mov     DS, AX
        call    isr_510
        pop     ES
        pop     DS
        pop     DI
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        pop     BP
        iret
ih510           ENDP

_PROG   ENDS
end
```

231928–71

82510 XMODEM Implementation (Continued)

82530 SCC TECHNICAL MANUAL

September 1986

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

LIST OF FIGURES

Table of Contents (Continued)

CHAPTER 1
GENERAL INFORMATION

1.0 INTRODUCTION

The Intel 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. The SCC functions as a serial-to-parallel, parallel-to-serial converter/controller. The SCC can be software-configured to satisfy a wide range of serial communications applications. The device contains new, sophisticated internal functions including on-chip baud rate generators, digital phase locked loops, various data encoding and decoding schemes, and crystal oscillators that dramatically reduce the need for external logic.

In addition, diagnostic capabilities - automatic echo and local loopback - allow the user to detect and isolate a failure in the network. They greatly improve the reliability and maintainability of the system.

The SCC handles Asynchronous formats, Synchronous byte-oriented protocols such as IBM Bisync, and Synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device supports virtually any serial data transfer application (Terminal, Personal Computer, Peripherals, Industrial Controller, Telecommunication system, etc.).

The 82530 can generate and check CRC codes in any Synchronous mode and can be programmed to check data integrity in various modes. The SCC also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

1.1 CAPABILITIES

Two independent full-duplex channels

Synchronous/Isosynchronous data rates:

- Up to 1 Megabit/second with 4 MHz clock rate
- Up to 250 Kbit/second with a 4 MHz clock rate (FM encoding)
- Up to 125 Kbit/second with a 4 MHz clock rate (NRZI encoding)

Asynchronous capabilities:

- 5, 6, 7, or 8 bits per character
- 1, $1\frac{1}{2}$, or 2 stop bits
- Odd or even parity
- Times 1, 16, 32, or 64 clock modes
- Break generation and detection
- Parity, overrun and framing error detection

Byte-oriented synchronous capabilities:

- Internal or external character synchronization
- 1 or 2 sync characters in separate registers
- Automatic sync character insertion and deletion
- Cyclic redundancy check (CRC) generation/detection
- 6- or 8-bit sync character

SDLC/HDLC capabilities:

- Abort sequence generation and checking
- Automatic zero insertion and deletion
- Automatic flag insertion between messages
- Address field recognition
- I-field residue handling
- CRC generation/detection
- SDLC loop mode with EOP recognition/loop entry and exit

Integrated features:

- NRZ, NRZI or FM encoding/decoding
- Manchester decoding
- Baud rate generator in each channel
- 8274 Non-Vectored Mode compatible
- Digital Phase-Locked Loop for clock recovery
- Crystal Oscillator

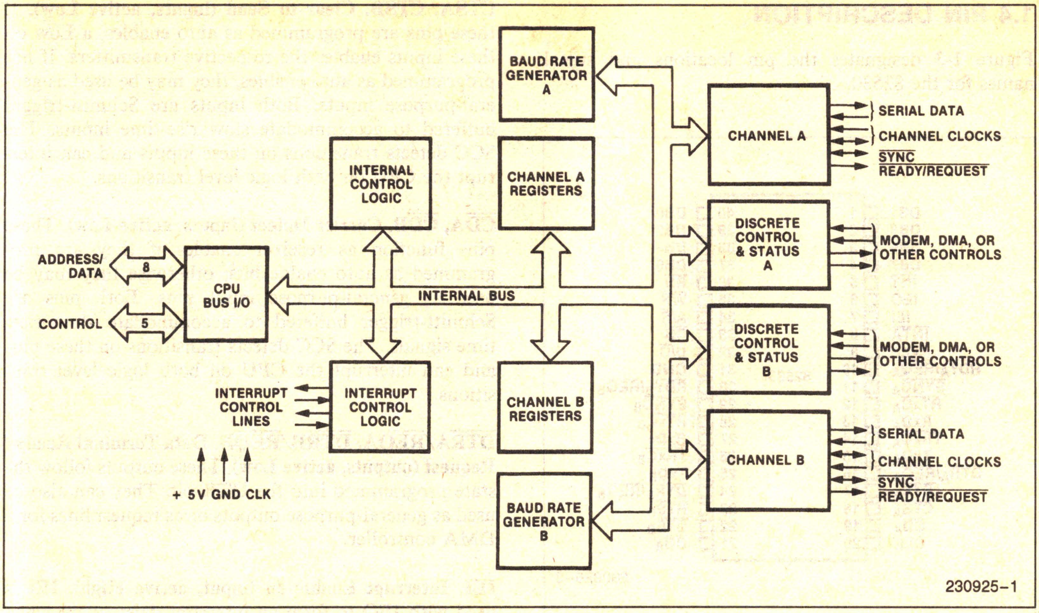


Figure 1-1. SCC Block Diagram

1.2 BLOCK DIAGRAM

Figure 1-1 is a block diagram of the SCC. Received data enters the receive data pins and follows one of several data paths, depending on the state of the control logic. The contents of the registers and the state of the external control pins establish the internal control logic. Transmitted data follows a similar pattern of control, register, and external pin definition.

1.3 PIN FUNCTIONS

The SCC pins are divided into seven functional groups: address/data, bus timing and reset, device control, interrupt, serial data (both channels), peripheral control (both channels), and clocks (both channels). Figure 1-2 shows the pins in each functional group.

The address/data group consists of the bidirectional lines used to transfer data between the CPU and the SCC. The direction of these lines depends on whether the SCC is selected and whether the operation is a read or a write.

The timing and control groups designate the type of transaction to occur and when this transaction will occur. The remaining groups are divided into Channel A and Channel B data groups for serial data (transmit or receive), peripheral control (such as DMA or modem), and the input and output lines for the receive and transmit clocks.

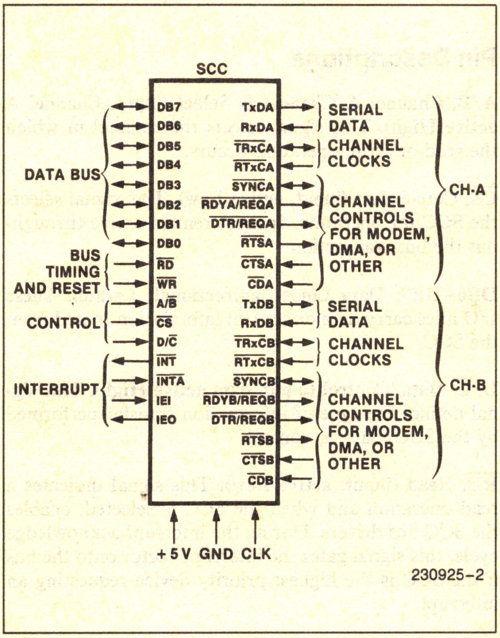
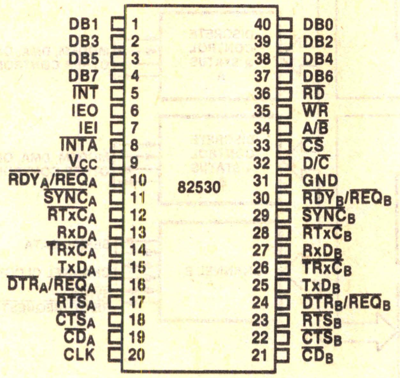


Figure 1-2. Pin Functions

1.4 PIN DESCRIPTION

Figure 1-3 designates the pin locations and signal names for the 82530.



230925-3

Figure 1-3. Pin Designation

Pin Descriptions

A/$\overline{\text{B}}$. Channel A/Channel B Select (input, Channel A active High). This signal selects the channel in which the read or write operation occurs.

$\overline{\text{CS}}$. Chip-Select (input, active Low). This signal selects the SCC for operation. It must remain active throughout the bus transaction.

DB0–DB7. Data Lines (bidirectional, 3-state). These I/O lines carry data or control information to and from the SCC.

D/$\overline{\text{C}}$. Data/Control (input, data active High). This signal defines the type of information transfer performed by the SCC: data or control.

$\overline{\text{RD}}$. Read (input, active Low). This signal indicates a read operation and when the SCC is selected, enables the SCC bus drivers. During the interrupt acknowledge cycle, this signal gates the interrupt vector onto the bus if the SCC is the highest priority device requesting an interrupt.

$\overline{\text{WR}}$. Write (input, active Low). When the SCC is selected, this signal indicates a write operation. The coincidence of $\overline{\text{RD}}$ and $\overline{\text{WR}}$ is interpreted as a reset.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$. Clear to Send (inputs, active Low). If these pins are programmed as auto enables, a Low on these inputs enables the respective transmitters. If not programmed as auto enables, they may be used as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow rise-time inputs. The SCC detects transitions on these inputs and can interrupt the CPU on both logic level transitions.

$\overline{\text{CDA}}$, $\overline{\text{CDB}}$. Carrier Detect (inputs, active Low). These pins function as receiver enables if they are programmed as auto enable bits; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered to accommodate slow rise-time signals. The SCC detects transitions on these pins and can interrupt the CPU on both logic level transitions.

$\overline{\text{DTRA}}$/$\overline{\text{REQA}}$, $\overline{\text{DTRB}}$/$\overline{\text{REQB}}$. Data Terminal Ready/Request (outputs, active Low). These outputs follow the state programmed into the DTR bit. They can also be used as general-purpose outputs or as request lines for a DMA controller.

IEI. Interrupt Enable In (input, active High). IEI is used with IEO to form an interrupt daisy chain when there is more than one interrupt-driven device. A High on IEI indicates that no other higher priority device has an Interrupt Under Service (IUS) or is requesting an interrupt.

IEO. Interrupt Enable Out (output, active High). IEO is High only if IEI is High and the CPU is not servicing an SCC interrupt or the controller is not requesting an interrupt (interrupt acknowledge cycle only). IEO is connected to the next lower priority device's IEI input and thus inhibits interrupts from lower priority devices.

$\overline{\text{INTA}}$. Interrupt Acknowledge (input, active Low). This signal indicates an active interrupt acknowledge cycle. During this cycle, the interrupt daisy chain settles. When $\overline{\text{RD}}$ becomes active, the SCC places an interrupt vector on the data bus (if IEI is High). $\overline{\text{INTA}}$ is latched by the rising edge of $\overline{\text{RD}}$ or CLK.

$\overline{\text{INT}}$. Interrupt Request (output, open-drain, active Low). This signal is activated when the SCC is requesting an interrupt.

CLK. Clock (input). This is the master clock used to synchronize internal signals. CLK is not required to have any phase relationship with the master system clock. CLK is a TTL level signal.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$. Request to Send (outputs, active Low). When the Request to Send (RTS) bit in Write Register 5 is set, the $\overline{\text{RTS}}$ signal goes Low. When the RTS bit is reset in the Asynchronous mode and auto

enable is on, the signal goes High after the transmitter is empty. In Synchronous mode or in Asynchronous mode with auto enable off, the RTS pins strictly follow the state of the RTS bit. Both pins can be used as general-purpose outputs.

RTxCA, RTxCB. Receive/Transmit Clocks (inputs, active Low). The functions of these pins are under program control. In each channel, RTxC may supply the receive clock, the transmit clock, the clock for the baud rate generator, or the clock for the Digital Phase-Locked Loop (refer to Section 4 for bit configurations). These pins can also be programmed for use with the respective SYNC pins as a crystal oscillator. The receive clock may be 1, 16, 32, or 64 times the data rate in asynchronous modes.

RxDA, RxDB. Receive Data (inputs, active High). These input signals receive serial data at standard TTL levels.

SYNCA, SYNCB. Synchronization (inputs/outputs, active Low). These pins can act as either inputs, outputs, or as part of the crystal oscillator circuit. In the Asynchronous Receive mode (crystal oscillator option not selected), these pins are inputs similar to CTS and CD. In this mode, transitions on these lines affect the state of the Sync/Hunt status bits in Read Register 0 (Figure 4-17) but have no other function.

In external Synchronization mode with the crystal oscillator not selected, these lines also act as inputs. In this mode, SYNC must be driven Low two receive clock cycles after the last bit in the sync character is received. Character assembly begins on the rising edge of the receive clock immediately preceding the activation of SYNC.

In the Internal Synchronization mode (Monosync and Bisync) with the crystal oscillator not selected, these pins act as outputs and are active only during the part of the receive clock cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync character is recognized (regardless of character boundaries). In SDLC mode, these pins act as outputs and are valid on receipt of a flag.

TRxCA, TRxCB. Transmit/Receive Clocks (inputs or outputs, active Low). The functions of these pins are under program control. TRxC may supply the receive clock or the transmit clock in the Input mode or supply the output of the Digital Phase-Locked Loop, the crystal oscillator, the baud rate generator, or the transmit clock in the output mode. (Refer to Section 4 for bit configuration.)

TxDA, TxDB. Transmit Data (outputs, active High). This output signal transmits serial data at standard TTL levels.

RDYA/REQA, RDYB/REQB. Ready/Request, (outputs, open-drain when programmed for Ready function, driven High or Low when programmed for a Request function). These dual-purpose outputs can be programmed as Request lines for a DMA controller or as Ready lines to synchronize the CPU to the SCC data rate. The reset state is Ready.

CHAPTER 2
ARCHITECTURE

2.0 INTRODUCTION

The SCC internal structure provides all the interrupt and control logic necessary to interface with nonmultiplexed buses. Interface logic is also provided to monitor modem or peripheral control inputs and outputs. All of the control signals are general purpose and can be applied to various peripheral devices as well as used for modem control.

The center for data activity revolves around the internal read and write registers. The programming of these registers provides the SCC with a functional "personality", i.e., register values can be assigned before or during program sequencing to determine how the SCC will establish a given communication protocol.

2.1 REGISTER FUNCTIONS

All modes of communication are established by the bit values of the write registers. As data is received or transmitted, read register values may change. These changed values can promote software action or internal hardware action for further register changes.

The register set for each channel includes 14 write registers and seven read registers. Ten write registers are used for control, two for sync character generation, and two for baud rate generation. The remaining two write registers are shared by both channels; one is used as the interrupt vector and one as the master interrupt control. Four read registers indicate status functions; two are used by the baud rate generator, one for the interrupt vector, one for the receiver buffer, and one for reading the interrupt pending bits.

Table 2-1 lists the assigned functions for each read and write register. The SCC contains only one WR2 (interrupt vector) and one WR9 (master interrupt control). Both registers are accessed and shared by either channel. Chapter 4 provides a detailed bit legend and description of each register.

Table 2-1. Register Set

Read Register	Function
RRO	Transmit/Receive buffer status, and External status
RR1	Special Receive Condition status, residue codes, error conditions
RR2	Modified (Channel B only) interrupt vector and Unmodified interrupt vector (Channel A only)
RR3	Interrupt Pending bits (Channel A only)
RR8	Receive buffer
RR10	Miscellaneous XMTR, RCVR status parameters
RR12	Lower byte of baud rate generator time constant
RR13	Upper byte of baud rate generator time constant
RR15	External/Status interrupt control information

Write Register	Function
WRO	Command Register, Register Pointers, CRC initialization, resets for various modes
WR1	Interrupt conditions, Ready/DMA request control
WR2	Interrupt vector (access through either channel)
WR3	Receive/Control parameters, number of bits per character, Rx CRC enable
WR4	Transmit/Receive miscellaneous parameters and modes, clock rate, number of sync characters, stop bits, parity
WR5	Transmit parameters and controls, number of Tx bits per character, Tx CRC enable
WR6	Sync character (1st byte) or SDLC address field
WR7	Sync character (2nd byte) or SDLC flag
WR8	Transmit buffer
WR9	Master interrupt control and reset (accessed through either channel), reset bits, control interrupt daisy chain
WR10	Miscellaneous transmitter/receiver control bits, NRZI, NRZ, FM encoding, CRC reset
WR11	Clock mode control, source of Rx and Tx clocks
WR12	Lower byte of baud rate generator time constant
WR13	Upper byte of baud rate generator time constant
WR14	Miscellaneous control bits: baud rate generator, Phase-Locked Loop control, auto echo, local loopback
WR15	External/Status interrupt control information-control external conditions causing interrupts

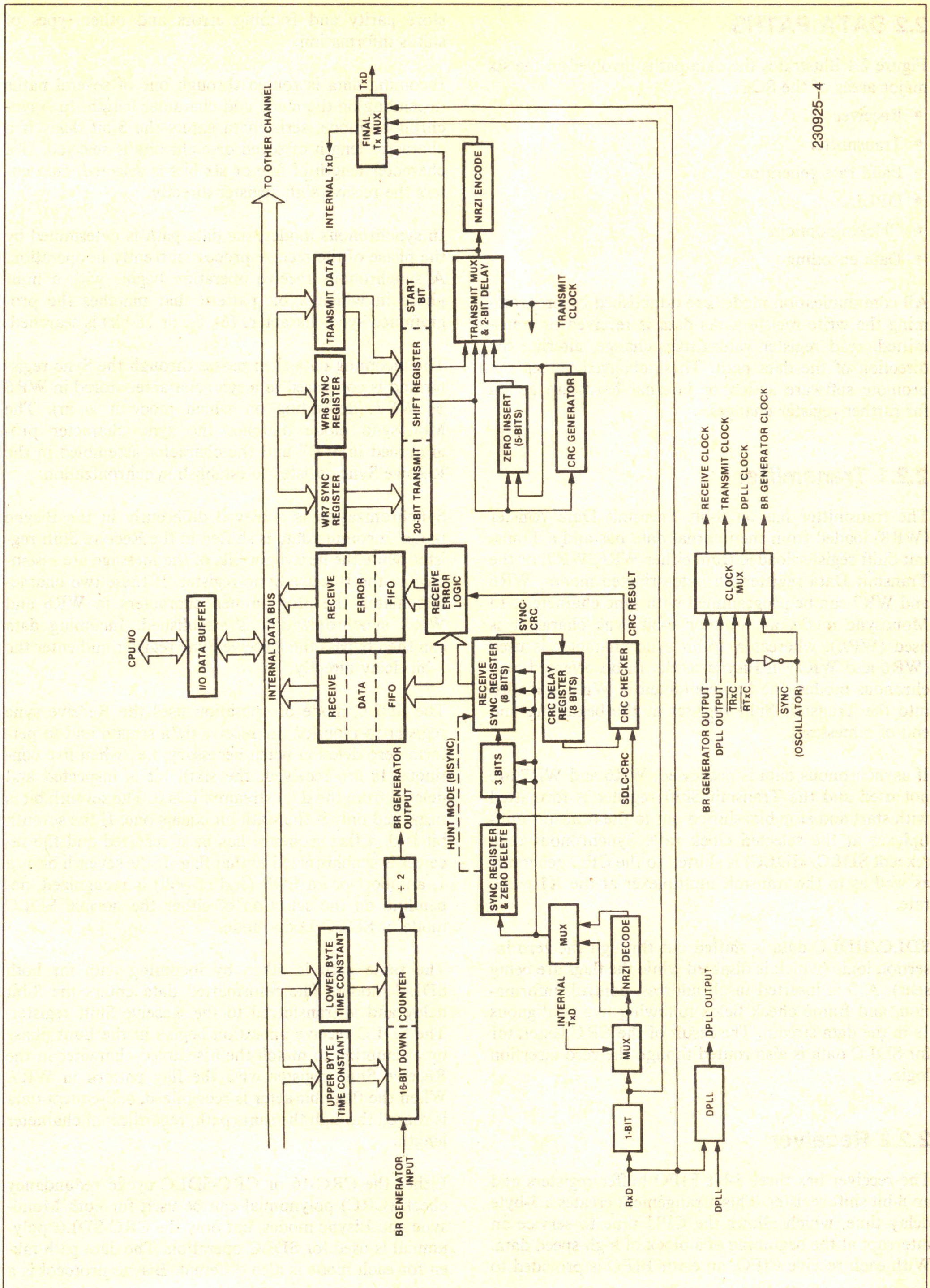


Figure 2-1. Data Paths

2.2 DATA PATHS

Figure 2.1 illustrates the data paths involved in the six major areas of the SCC:

- Receiver
- Transmitter
- Baud rate generator
- DPLL
- Clocking options
- Data encoding

All communication modes are established by programming the write registers. As data is received or transmitted, read register values may change, altering the direction of the data path. These changed values can promote software action or internal hardware action for further register changes.

2.2.1 Transmitter

The transmitter has an 8-bit Transmit Data register (WR8) loaded from the internal data bus and a Transmit Shift register loaded from either WR6, WR7, or the Transmit Data register. In byte-oriented modes, WR6 and WR7 can be programmed with sync characters. In Monosync mode, an 8-bit or 6-bit sync character is used (WR6), whereas a 16-bit sync character is used (WR6 and WR7) in Bisync mode. In bit-oriented synchronous modes, the flag contained in WR7 is loaded into the Transmit Shift register at the beginning and end of a message.

If asynchronous data is processed, WR6 and WR7 are not used and the Transmit Shift register is formatted with start and stop bits shifted out to the transmit multiplexer at the selected clock rate. Synchronous data (except SDLC/HDLC) is shifted to the CRC generator as well as to the transmit multiplexer at the X1 clock rate.

SDLC/HDLC data is shifted out through the zero insertion logic (which is disabled while the flags are being sent). A 0 is inserted in all address, control, information, and frame check fields following five contiguous 1s in the data stream. The result of the CRC generator for SDLC data is also routed though the zero insertion logic.

2.2.2 Receiver

The receiver has three 8-bit FIFO buffer registers and an 8-bit shift register. This arrangement creates a 3-byte delay time, which allows the CPU time to service an interrupt at the beginning of a block of high-speed data. With each receive FIFO, an error FIFO is provided to store parity and framing errors and other types of status information.

Incoming data is routed through one of several paths depending on the mode and character length. In Asynchronous mode, serial data enters the 3-bit delay if a character length of seven or eight bits is selected. If a character length of five or six bits is selected, data enters the receive shift register directly.

In synchronous modes, the data path is determined by the phase of the receive process currently in operation. A synchronous receive operation begins with a hunt phase in which a bit pattern that matches the programmed sync characters (6-, 8-, or 16-bit) is searched.

The incoming data then passes through the Sync register and is compared to a sync character stored in WR6 or WR7 (depending on which mode it is in). The Monosync mode matches the sync character programmed in WR7 and the character assembled in the Receive Sync register to establish synchronization.

Synchronization is achieved differently in the Bisync mode. Incoming data is shifted to the Receive Shift register while the next eight bits of the message are assembled in the Receive Sync register. If these two characters match the programmed characters in WR6 and WR7, synchronization is established. Incoming data can then bypass the Receive Sync register and enter the 3-bit delay directly.

The SDLC mode of operation uses the Receive sync register to monitor the receive data stream and to perform zero deletion when necessary; i.e., when five continous 1s are received, the sixth bit is inspected and deleted from the data stream if it is 0. The seventh bit is inspected only if the sixth bit equals one. If the seventh bit is 0, a flag sequence has been received and the receiver is synchronized to that flag. If the seventh bit is a 1, an abort or an EOP (end of poll) is recognized, depending on the selection of either the normal SDLC mode or SDLC Loop mode.

The same path is taken by incoming data for both SDLC modes. The reformatted data enters the 3-bit delay and is transferred to the Receive Shift register. The SDLC receive operation begins in the hunt phase by attempting to match the assembled character in the Receive Shift register with the flag pattern in WR7. When the flag character is recognized, subsequent data is routed through the same path, regardless of character length.

Either the CRC-16 or CRC-SDLC cyclic redundancy check (CRC) polynomial can be used for both Monosync and Bisync modes, but only the CRC-SDLC polynomial is used for SDLC operation. The data path taken for each mode is also different. Bisync protocol is a

byte-oriented operation that requires the CPU to decide whether or not a data character is to be included in CRC calculation. An 8-bit delay in all synchronous modes except SDLC is allowed for this process. In SDLC mode, all bytes are included in the CRC calculation.

2.3 DATA COMMUNICATIONS CAPABILITIES

SCC logic handles all asynchronous, byte-oriented synchronous, and bit-oriented synchronous modes of operation. The following section briefly describes asynchronous, synchronous, and SDLC modes of communication.

2.3.1 Asynchronous

Figure 2-2 represents a typical asynchronous message format using one start bit, seven data bits, one parity bit, and one stop bit. A start bit is a High-to-Low transition detected by an asynchronous receiver and is actually an information bit notifying the receiver of an incoming message.

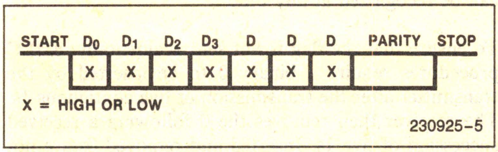


Figure 2-2. Asynchronous Message Format

The start bit also initiates a clock circuit to provide latching pulses during expected data bit intervals. The parity bit is provided for error checking and rests in a resultant state (odd or even) depending on an accumulated 1's state count of the data bit. The parity bit is calculated in both the receiver and the transmitter; the two results are compared to ensure that the expected and the actual bit values match.

The stop bit returns the message unit to the quiescent marking state; i.e., a constant high state condition lasts until the next High-to-Low start bit indicates an incoming data byte. During reception, the start and stop bits are stripped away and checked for errors, leaving only the working data for CPU interaction. The number of selected bits for each asynchronous function may vary in the transmitter and the receiver.

2.3.2 Monosync Mode

Monosync and Bisync modes require clocking information to be transmitted along with the data by a method of encoding data that contains clocking information, or by a modem to encode or decode clock information in the modulation process.

Start and stop bits are not required in synchronous modes. All bits are used to transmit data, which eliminates the "waste" characteristic of asynchronous communication.

Figure 2-3 shows the character format for synchronous transmission. For example, bits 1–8 might be one character and bits 9–13 part of another character; or bit 1 might be part of one character, bits 2–9 part of a second character, and bits 10–13 part of a third character. The framing (where each character begins) of each character is accomplished by defining a synchronization character, commonly called a "sync character."

The CPU places the receiver in Hunt mode whenever transmission begins (or whenever a data dropout has occurred and the hardware determines that resynchronization is necessary). In Hunt mode, the receiver shifts a bit into the Receive Shift register and compares the contents of the Receive Shift register with the sync character (stored in another register), repeating the process until a match occurs. When a match occurs, the receiver begins transferring bytes to the receive FIFO.

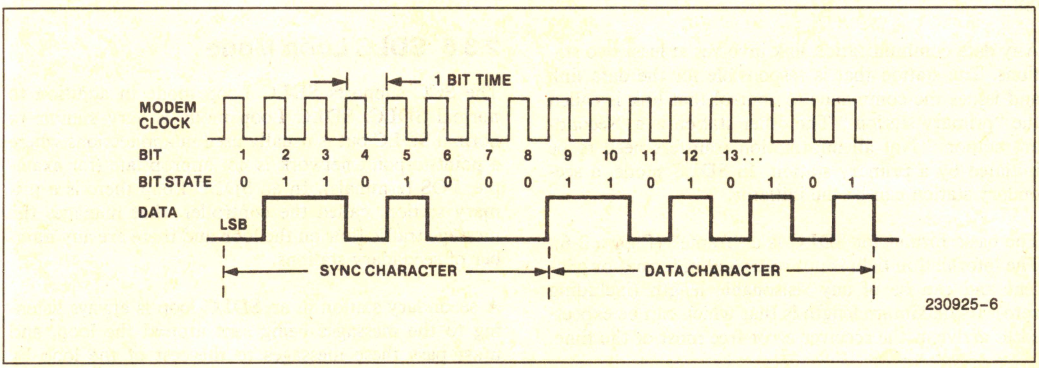


Figure 2-3. Monosync Data Character Format

2.3.3 Bisynchronous Mode

The Bisync mode of operation (Figure 2-4) is similar to the Monosync mode, except that two sync characters are provided instead of one. Bisync attempts a more structured approach to synchronization through the use of special characters as message "headers" or "trailers" (refer to IBM's Bisync).

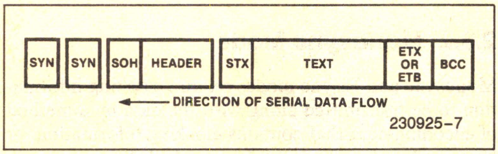


Figure 2-4. Bisynchronous Message Format

2.3.4 External Sync Mode

External Sync mode (Figure 2-5) eliminates the use of sync characters in the serial data stream by providing an external sync signal to mark the beginning of a data field; i.e., an external input pin (Sync) waits for an active state change to indicate the ensuing information field.

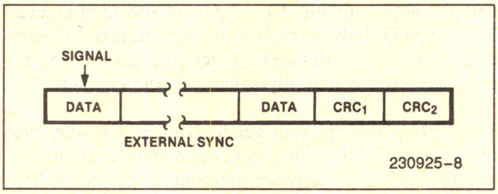


Figure 2-5. External Sync Format

2.3.5 SDLC Mode

Synchronous Data Link Control mode (SDLC) uses synchronization characters similar to Bisync and Monosync modes (such as flags and pad characters), but it is a bit-oriented protocol instead of byte-oriented protocol.

Any data communication link involves at least two stations. The station that is responsible for the data link and issues the commands to control that link is called the "primary station." The other station is a "secondary station." Not all information transfers need to be initiated by a primary station. In SDLC mode, a secondary station can be the initiator.

The basic format for SDLC is a "frame" (Figure 2-6). The information field is not restricted in format or content and can be of any reasonable length (including zero). Its maximum length is that which can be expected to arrive at the receiver error-free most of the time. Hence, the determination of maximum length is a function of communication channel error rate.

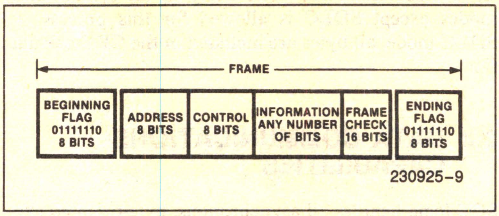


Figure 2-6. SDLC Message Format

The two flags that delineate the SDLC frame serve as reference points when positioning the address and control fields, and they initiate the transmission error check. The ending flag indicates to the receiving station that the 16 bits just received constitute the frame check. The ending flag could be followed by another frame, another flag, or an idle. This means that when two frames follow one another, the intervening flag may simultaneously be the ending flag of the first frame and the beginning flag of the next frame. Since the SDLC mode does not use characters of defined length, but rather works on a bit-by-bit basis, the 01111110 flag can be recognized at any time.

To ensure that the flag is not sent accidentally, SDLC procedures require a binary 0 to be inserted by the transmitter after the transmission of five contiguous 1s. The receiver then removes the 0 following a received succession of five 1s. Inserted and removed 0s are not included in the CRC calculation.

The address field is eight bits long and designates the number of the secondary station to which the commands or data from the primary station are sent. The control field is eight bits long and is used to initiate all SDLC activities (see section 3.7).

The SCC can also serve the high-level synchronous data link communication (HDLC) protocol, which is identical to SDLC except for differences in framing.

2.3.6 SDLC Loop Mode

The SCC supports SDLC Loop mode in addition to normal SDLC. SDLC Loop mode is very similar to normal SDLC but is usually used in applications where a point-to-point network is not appropriate (for example, POS terminals). In an SDLC Loop there is a primary station, called the controller, that manages the message traffic flow on the loop and there are any number of secondary stations.

A secondary station in an SDLC loop is always listening to the messages being sent around the loop, and must pass these messages to the rest of the loop by

retransmitting them with a one-bit-time delay. The secondary station can only place its own message on the loop at specific times. The controller signals that secondary stations may transmit messages by sending a special character, called an EOP (End Of Poll), around the loop. The EOP character is the bit pattern 11111110. Because of zero insertion during messages this bit pattern is unique and thus is easily recognized.

When a secondary station has a message to transmit and it recognizes an EOP on the line, the first thing that it does is to change the last 1 of the EOP to a 0 before transmitting it. This has the effect of turning the EOP into a Flag sequence. The secondary station now places its message on the loop and terminates its message with an EOP. Any secondary stations further down the loop with messages to transmit can then append their messages to the message of the first secondary station by the same process. Any secondary stations without messages to send merely echo the incoming messages and are prohibited from placing messages on the loop, except on recognizing an EOP.

There are also restrictions as to when and how a secondary station physically becomes part of the loop. A secondary station that has just powered up must monitor the loop, without the one-bit-time delay, until it recognizes an EOP. When an EOP is recognized the one-bit-time delay is switched on. This does not disturb the loop because the line is marking idle between the time that the controller sends the EOP and the time that it receives the EOP back. The secondary station that has gone on-loop cannot place a message on the loop until the next time that an EOP is issued by the controller. A secondary station goes off-loop in a similar manner. When given a command to go off-loop, the secondary station waits until the next EOP to remove the one-bit-time delay.

To operate the SCC in SDLC Loop mode, the SCC must first be programmed just as if normal SDLC were to be used. The baud rate generators should be set up and the Loop mode selected by writing the appropriate control word in WR11. Since WR11 also controls the clocking mode for the receiver and the transmitter, it may be useful to write the final clocking values in WR11 without selecting the SDLC Loop mode before any of the other registers are written. This allows faster startup by allowing the clocks to settle, but is not strictly necessary. If NRZI encoding and the DPLL are being used, these options should be selected before selecting SDLC Loop mode for the same reasons. The SCC is now waiting for the EOP so that it can go on loop. While waiting for the EOP, the SCC ties TxD to RxD with only the internal gate delays in the signal path. When the first EOP is recognized by the SCC, the Break/Abort/EOP bit is set in RR0, generating an External/Status interrupt (if so enabled). At the same time, the On-Loop bit in RR10 is set to indicate that the SCC is indeed on-loop, and a one-bit time delay is inserted in the TxD to the RxD path.

The SCC is now on-loop but cannot transmit a message until a flag and the next EOP are received. The requirement that a flag be received ensures that the SCC cannot erroneously send messages until the controller ends the current polling sequence and starts another one. If SDLC mode is deselected before this flag is received, the Break/Abort/EOP bit resets, generating another External/Status interrupt, and the SCC transmits messages in response to an EOP being received.

A secondary station on the loop is prohibited from transmitting a message during a polling sequence unless it captures the line at the moment the EOP passes by. The SCC does this automatically. If the CPU in the secondary station with SCC needs to transmit a message, the Go-Active-On-Poll bit in WR10 must be set. If this bit is set when the EOP is detected, the SCC changes the EOP to a flag and starts sending another flag. The EOP is reported in the Break/Abort/EOP bit in RR0 and the CPU should write its data bytes to the SCC, just as in normal SDLC frame transmission. When the frame is complete and CRC has been sent, the SCC closes with a flag and reverts to One-Bit-Delay mode. The last zero of the flag, along with the marking line echoed from the RxD pin, form an EOP for secondary stations further down the loop. If the Go-Active-On-Poll bit is not set at the time the EOP passes by, the SCC cannot send a message until a flag (terminating the current polling sequence) and another EOP are received. While the SCC is actually transmitting a message, the Loop-Sending bit in RR10 is set to indicate this.

If SDLC loop is deselected, the SCC is designed to exit from the loop gracefully. When SDLC Loop mode is deselected by writing to WR10, the SCC waits until the next polling cycle to remove the one-bit time delay. If a polling cycle is in progress at the time the command is written, the SCC finishes sending any message that it may be transmitting, ends with an EOP, and disconnects TxD from RxD. If no message was in progress, the SCC immediately disconnects TxD from RxD. If a polling cycle is not in progress at the time the command is given, the SCC waits until an EOP is recognized to disconnect TxD from RxD. To ensure proper loop operation after the SCC goes off the loop, and until the external relays take the SCC completely out of the loop, the SCC should be programmed for Mark idle instead of Flag idle. When the SCC goes off the loop, the On-Loop bit is reset.

The SCC allows the user the option of using NRZI in SDLC Loop mode by programming WR10 appropriately. With NRZI encoding, the outputs of secondary stations in the loop may be inverted from their inputs because of messages that they have transmitted. Removing the stations from the loop (removing the one-

bit time delay) may cause problems further down the loop because of extraneous transitions on the line. The SCC removes this problem by making transparent adjustments at the end of each frame it sends in response to an EOP. A response frame from the SCC is terminated by a flag and an EOP. Normally, the flag and the EOP share a zero, but if such sharing would cause the RxD and TxD pins to be of opposite polarity after the EOP, the SCC adds another zero between the flag and the EOP. This causes an extra line transition so that RxD and TxD are identical after the EOP is sent. This extra zero is completely transparent because it only means that the flag and the EOP no longer share a zero. All that a proper loop exit needs, therefore, is the removal of the one-bit time delay.

2.4 I/O CAPABILITIES

The SCC can work with three basic forms of I/O operations: polling, interrupts, and block transfer. All three I/O types involve register manipulation during initialization and data transfer.

2.4.1 Polling

During a polling sequence, the status of Read Register 0 is examined in each channel. This register indicates whether or not a receive or transmit data transfer is needed and whether or not any special conditions are present, e.g., errors.

This method of I/O transfer avoids interrupts. All interrupt functions must be disabled in order to operate the device in a polled environment. With no interrupts enabled, this mode of operation must initiate a read cycle of Read Register 0 to detect an incoming character before jumping to a data handler routine.

2.4.2 Interrupts

The SCC provides interrupt capability through the use of pins and a hardware scheme that increases the transfer speed of serial data. Whenever the interrupt ($\overline{\text{INT}}$) pin is active, the SCC is ready to transfer data.

Read and write registers are programmed so that an interrupt vector points to an interrupt service routine. The interrupt vector can also be modified to reflect various status conditions. Therefore, as many as eight different interrupt routines can be referenced.

Transmit interrupts, receive interrupts, and external/status interrupts are the main sources of interrupts. Each interrupt source is enabled under program control, with channel A having a higher priority than channel B and with receiver, transmit, and external/status interrupts prioritized respectively within each channel. (Section 3.3 provides a detailed description of the interrupt scheme and the various interrupt types.)

2.4.3 Block Transfer

The SCC provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers. The Block Transfer mode uses the $\overline{\text{RDY}}/\overline{\text{REQ}}$ output in conjunction with the Ready/Request bits in Write Register 1. The $\overline{\text{RDY}}/\overline{\text{REQ}}$ output can be defined by software as a $\overline{\text{READY}}$ line in the CPU Block Transfer mode or as a $\overline{\text{REQUEST}}$ line in the DMA Block Transfer mode.

To a DMA controller, the $\overline{\text{REQUEST}}$ output indicates that the SCC is ready to transfer data to or from memory. To the CPU, the $\overline{\text{RDY}}$ output indicates that the SCC is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle. (Section 3.4 describes the registers used in block transfers.)

2.5 SUPPORT CIRCUITRY

The SCC incorporates additional circuitry to aid serial communications. The designer can select an internal baud rate generator, select the frequency, and program the output to one of several circuits contained within the SCC. The SCC can be programmed to encode and decode in several standard formats. In addition, various clocking options can be selected for the DPLL, the baud rate generator, the receiver and transmitter.

2.5.1 Baud Rate Generator

Each channel in the SCC contains a programmable baud rate generator. Each generator consists of two 8-bit, time-constant registers forming a 16-bit time constant, a 16-bit down counter, and a flip-flop on the output that makes the output a square wave. On startup, the flip-flop on the output is set High so that it starts in a known state, the value in the time-constant register is loaded into the counter, and the counter begins counting down. When a count of zero is reached, the output of the baud rate generator toggles, the value in the time-constant register is loaded into the counter, and the process starts over. The time constant can be changed at any time, but the new value does not take effect until the next load of the counter.

No attempt is made to synchronize the loading of a new time constant with the clock used to drive the generator. When the time constant is to be changed, the generator is stopped by writing to an enable bit in WR14. This ensures the loading of a correct time constant.

If neither the transmit clock nor the receive clock are programmed to come from the TRXC pin, the output

of the baud rate generator may be made available for external use on the TRXC pin. Section 3.8 presents the formula for determining the time constant for a given rate.

2.5.2 Digital Phase-Locked Loop (DPLL)

The SCC contains a Digital Phase-Locked Loop that can be used to recover clock information from a data stream with NRZI or FM coding. The DPLL is driven by a clock nominally 32 (NRZI) or 16 (FM) times the data rate. The DPLL uses this clock, along with the data stream, to construct a receive clock for the data. This clock can then be used as the SCC receive clock, the transmit clock, or both. Section 3.9 details the clock recovery for each of the different forms of encoding.

2.5.3 Clocking Options

The SCC can select several clock sources for internal and external use. Write Register 11 is the Clock Mode Control register for both the receive and transmit clocks. It determines the type of signal on the SYNC and $\overline{RTxC}$ pins and the direction of the $\overline{TRxC}$ pin.

Write Register 11 also controls the output of the baud rate generator, the DPLL output, and the selection of either a TTL or an XTAL output for the $\overline{RTxC}$ pin. (Section 3.10 gives a detailed description of the clocking options.)

2.5.4 Data Encoding

Figure 2-7 illustrates the four encoding methods used by the SCC. In NRZ encoding, a 1 is represented by a High level and a 0 is represented by a Low level. In NRZI encoding, a 1 is represented by no change in level and a 0 is represented by a change in level. In FM1 (more properly, biphase mark), a transition occurs at the beginning of every bit cell. A 1 is represented by an additional transition at the center of the bit cell and a 0 is represented by the absence of a transition at the center of the bit cell. In FM0 (more properly, biphase space), a transition occurs at the beginning of every bit cell. A 0 is represented by an additional transition at the center of the bit cell and a 1 is represented by the absence of a transition at the center of the bit cell.

In addition to these four methods, the SCC can be used to decode Manchester (biphase level) data by using the DPLL in the FM mode and programming the receiver for NRZ data. Manchester encoding always produces a transition at the center of the bit cell. If the transition is Low to High, the bit is 0. If the transition is High to Low, the bit is 1.

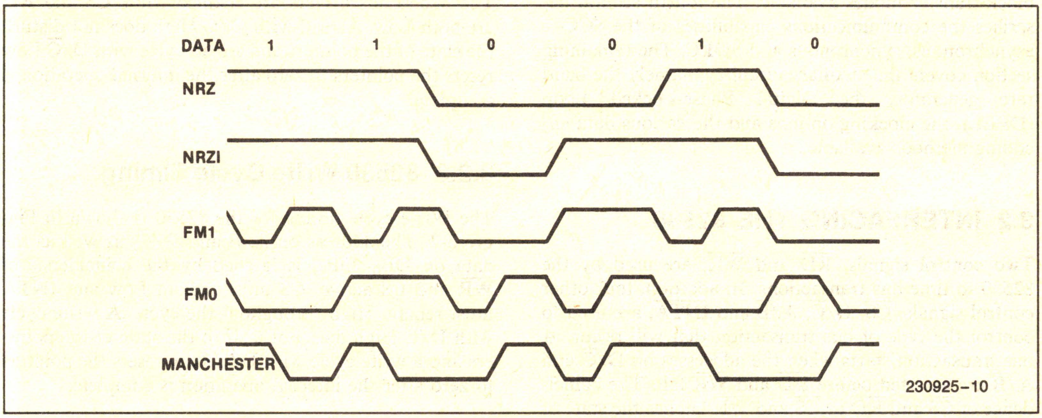


Figure 2-7. Data Encoding Methods

CHAPTER 3
FUNCTIONAL DESCRIPTION

3.0 INTRODUCTION

This chapter is a detailed description of the operation of the SCC. It is broken up by logical function so that functions specific to a particular mode are grouped together. This leads to some repetition but makes this chapter easier to use. Some programming data is included, but this chapter is neither a set of register descriptions nor an application note. Although the pertinent control bits are identified for each mode, no initialization routines are present. Typical initialization routines are not included because the SCC has so many permutations of operating modes and is suitable for so many applications.

3.1 ORGANIZATION

This chapter is organized into several sections, each covering a specific mode or function. The first sections cover the details of interfacing the 82530 to a system. The general timing requirements for both devices are described but the respective Product Specifications must be referred to for specific A.C. numbers. The third section covers the interrupt operation of the 82530, and the following section describes the various block-transfer modes available. The fourth section describes the communications capabilities of the SCC—asynchronous, synchronous and SDLC. The remaining section covers the auxiliary circuitry, namely the baud rate generator, the Digital Phase-Locked Loop (DPLL), the clocking options and the various data encoding methods available.

3.2 INTERFACING THE 82530

Two control signals, $\overline{RD}$ and $\overline{WR}$, are used by the 82530 to time bus transactions. In addition, four other control signals, $\overline{CS}$, D/$\overline{C}$, A/$\overline{B}$ and $\overline{INTA}$, are used to control the type of bus transaction that will occur. A bus transaction starts when the addresses on D/$\overline{C}$ and A/$\overline{B}$ are asserted before $\overline{RD}$ and $\overline{WR}$ fall. The coincidence of $\overline{CS}$ and $\overline{RD}$ or $\overline{CS}$ and $\overline{WR}$ latches the state of D/$\overline{C}$ and A/$\overline{B}$ and starts the internal operation. The $\overline{INTA}$ signal must have been previously sampled High by a rising edge of CLK for a read or write cycle to occur. In addition to sampling $\overline{INTA}$, CLK is used by the interrupt section to set the IP bits. The 82530 generates internal control signals in response to a register access. Since $\overline{RD}$ and $\overline{WR}$ have no phase relationship with CLK, the circuitry generating these internal control signals must provide time for metastable conditions to disappear. This gives rise to a recovery time related to CLK. This recovery time applies only between transactions involving the 82530, and any intervening transactions are ignored. The recovery time is six CLK cycles, measured from the falling edge of $\overline{RD}$ or $\overline{WR}$ in the case of a read or write of any register other than RR8 or WR8. In the case of a read of RR8 or a write to WR8 the recovery time is measured from the rising edge of $\overline{RD}$ or $\overline{WR}$.

3.2.1 82530 Read Cycle Timing

The Read cycle timing for the 82530 is shown in Figure 3-1. The address on A/$\overline{B}$ and D/$\overline{C}$ is latched by the coincidence of $\overline{RD}$ and $\overline{CS}$ active. $\overline{CS}$ must remain Low and $\overline{INTA}$ must remain High throughout the cycle. The 82530 bus drivers are enabled while $\overline{CS}$ and $\overline{RD}$ are both Low. A read with D/$\overline{C}$ High does not disturb the state of the pointers and a read cycle with D/$\overline{C}$ Low resets the pointers to zero after the internal operation is complete.

3.2.2 82530 Write Cycle Timing

The Write cycle timing for the 82530 is shown in Figure 3-2. The address on A/$\overline{B}$ and D/$\overline{C}$, as well as the data on DB_0–DB_7, is latched by the coincidence of $\overline{WR}$ and $\overline{CS}$ active. $\overline{CS}$ must remain Low and $\overline{INTA}$ must remain High throughout the cycle. A write cycle with D/$\overline{C}$ High does not disturb the state of the pointers and a write cycle with D/$\overline{C}$ Low resets the pointers to zero after the internal operation is complete.

3.2.3 82530 Interrupt Acknowledge Cycle Timing

The Interrupt Acknowledge cycle timing for the 82530 is shown in Figure 3-3. $\overline{\text{INTA}}$ is sampled by the rising edge of CLK, and when first sampled Low, the 82530 begins an Interrupt Acknowledge cycle. While $\overline{\text{INTA}}$ is Low, the state of A/$\overline{\text{B}}$, D/$\overline{\text{C}}$ and $\overline{\text{WR}}$ are ignored. Between the time $\overline{\text{INTA}}$ is first sampled Low and the time $\overline{\text{RD}}$ falls, the internal and external IEI/IEO daisy chains settle. If there is an interrupt pending in the 82530 and IEI is High when $\overline{\text{RD}}$ falls, the Interrupt Acknowledge cycle was intended for the 82530. This being the case, the 82530 sets the appropriate Interrupt Under Service latch, as well as placing an interrupt vector on DB_0–DB_7. The vector response is optional in the 82530 and there is no cycle time associated with the Interrupt Acknowledge cycle. There should be only one $\overline{\text{RD}}$ per Acknowledge cycle.

3.2.4 82530 Register Access

The registers in the 82530 are accessed in a two-step process, using a Register Pointer to perform the addressing. To access a particular register, the pointer bits must be set by writing to WR0. The pointer bits may be written in either channel because only one set exists in the 82530. After the pointer bits are set, the next read or write cycle of the 82530 having D/$\overline{\text{C}}$ Low will access

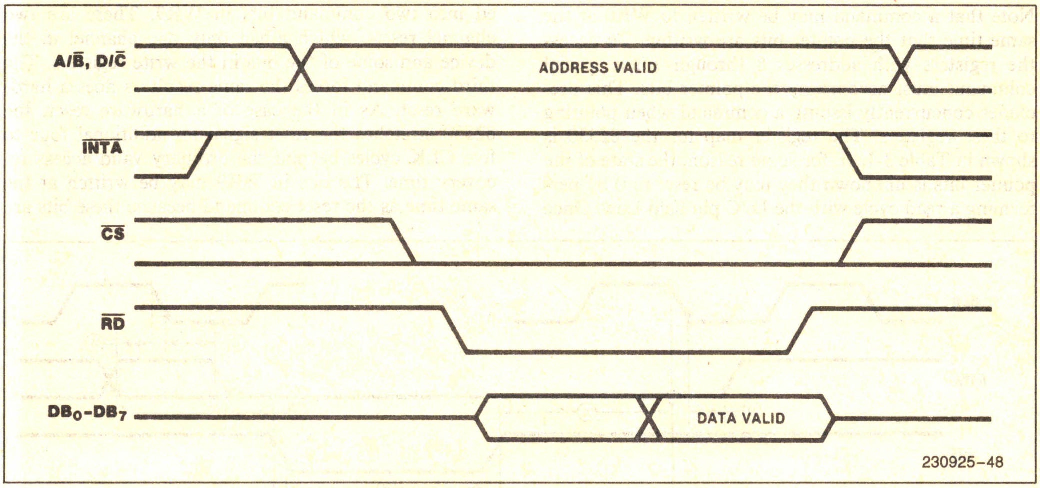


Figure 3-1. 82530 Read Cycle Timing

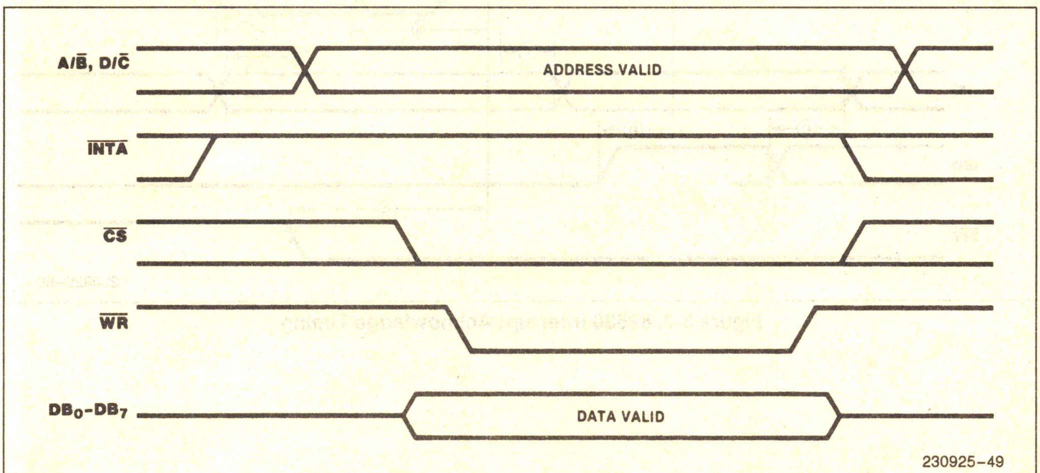


Figure 3-2. 82530 Write Cycle Timing

the desired register. At the conclusion of this read or write cycle the pointer bits are reset to 0s, so that the next control write will be to the pointers in WR0. A read of RR8—the receive data buffer, or a write to WR8—the transmit data buffer, may either be done in this fashion or by accessing the 82530 with the D/$\overline{C}$ pin High. A read or write with D/$\overline{C}$ High accesses the data registers directly, and independently, of the state of the pointer bits. This allows single-cycle access to the data registers and does not disturb the pointer bits. The fact that the pointer bits are reset to 0, unless explicitly set otherwise, means that WR0 and RR0 may also be accessed in a single cycle. That is, it is not necessary to write the pointer bits with 0 before accessing WR0 or RR0. There are three pointer bits in WR0, and these allow access to the registers with addresses 0 through 7. Note that a command may be written to WR0 at the same time that the pointer bits are written. To access the registers with addresses 8 through 15, a special command must accompany the pointer bits. This precludes concurrently issuing a command when pointing to these registers. The register map for the 82530 is shown in Table 3-1. If, for some reason, the state of the pointer bits is unknown they may be reset to 0 by performing a read cycle with the D/$\overline{C}$ pin held Low. Once the pointer bits have been set, the desired channel is selected by the state of the A/$\overline{B}$ pin during the actual read or write of the desired register.

3.2.5 82530 Reset

The 82530 may be reset by either hardware or software. Hardware reset occurs when $\overline{RD}$ and $\overline{WR}$ are both Low, at the same time, which is normally an illegal condition. As long as both $\overline{RD}$ and $\overline{WR}$ are Low, the 82530 recognizes the reset condition. Once this condition is removed, however, the reset condition is asserted internally for an additional four to five CLK cycles. During this time any attempt to access the 82530 will be ignored. The 82530 has three software resets, encoded into two command bits in WR9. There are two channel resets, which affect only one channel in the device and some of the bits in the write registers. The third command forces the same result as does a hardware reset. As in the case of a hardware reset, the 82530 stretches the reset signal an additional four to five CLK cycles beyond the ordinary valid access recovery time. The bits in WR9 may be written at the same time as the reset command because these bits are

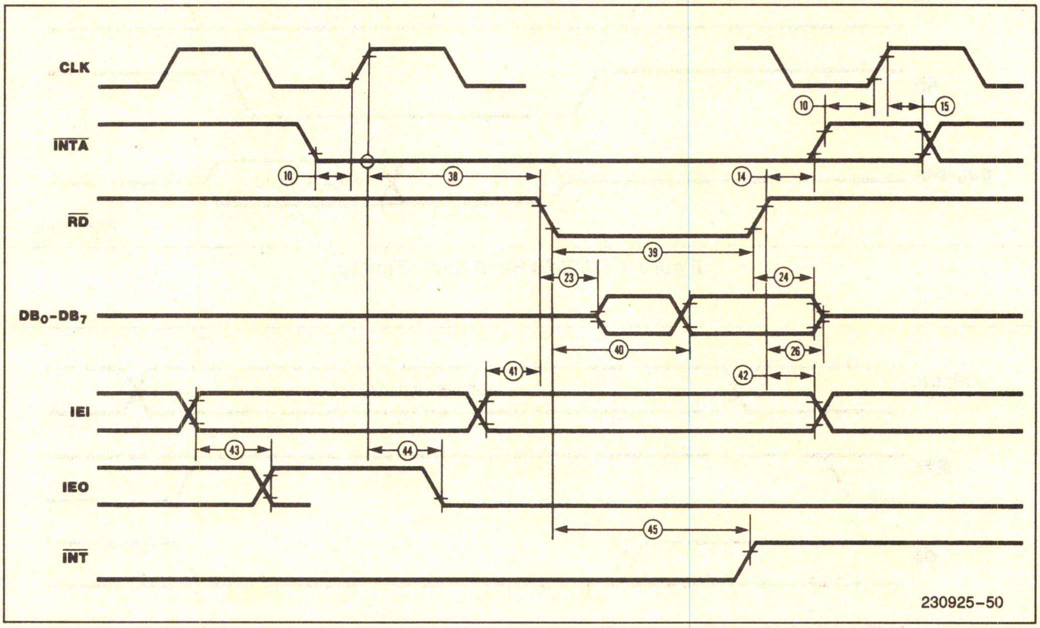


Figure 3-3. 82530 Interrupt Acknowledge Timing

affected only by a hardware reset. The reset values of the various registers are shown in Figure 3-4.

Table 3-1. 82530 Register Map

A/$\overline{B}$	PNT_2	PNT_1	PNT_0	WRITE	READ
0	0	0	0	WR0B	RR0B
0	0	0	1	WR1B	RR1B
0	0	1	0	WR2	RR2B
0	0	1	1	WR3B	RR3B
0	1	0	0	WR4B	(RR0B)
0	1	0	1	WR5B	(RR1B)
0	1	1	0	WR6B	(RR2B)
0	1	1	1	WR7B	(RR3B)
1	0	0	0	WR0A	RR0A
1	0	0	1	WR1A	RR1A
1	0	1	0	WR2	RR2A
1	0	1	1	WR3A	RR3A
1	1	0	0	WR4A	(RR0A)
1	1	0	1	WR5A	(RR1A)
1	1	1	0	WR6A	(RR2A)
1	1	1	1	WR7A	(RR3A)
With the Point High command:					
0	0	0	0	WR8B	RR8B
0	0	0	1	WR9	RR13B
0	0	1	0	WR10B	RR10B
0	0	1	1	WR11B	(RR15B)
0	1	0	0	WR12B	RR12B
0	1	0	1	WR13B	RR13B
0	1	1	0	WR14B	(RR10B)
0	1	1	1	WR15B	RR15B
1	0	0	0	WR8A	RR8A
1	0	0	1	WR9	(RR13A)
1	0	1	0	WR10A	RR10A
1	0	1	1	WR11A	(RR15A)
1	1	0	0	WR12A	RR12A
1	1	0	1	WR13A	RR13A
1	1	1	0	WR14A	RR15A
1	1	1	1	WR15A	RR15A

3.3 INTERRUPTS

Each channel in the SCC contains three sources of interrupts, making a total of six in all. These three sources of interrupts are the receiver, the transmitter, and external/status conditions. In addition, there are usually several conditions that may cause any of these interrupts. Each source of interrupt in the SCC has three control/status bits associated with it. They are Interrupt Enable (IE), Interrupt Pending (IP), and Interrupt Under Service (IUS). (See Figure 3-5.) The IE bit is written by the processor and serves to control interrupt requests from the SCC. If the IE bit is set for a given source of interrupt, then that source may cause an interrupt request when all of the necessary conditions are met. If the IE bit is reset, no interrupt request will be generated by that source. The IE bits are write-only in the SCC. The IP bit for a given source of interrupt may be set by the presence of an interrupt condition in the SCC and is reset directly by the processor, or indirectly by some action that the processor may take. If the corresponding IE bit is not set, the IP for that source of interrupt will never be set. The IP bits in the SCC may be considered to be read-only in RR3A. The IUS bits are completely hidden from the processor's view. In the Vectored Mode (WR9 bit D5 = 0), an IUS is set during an interrupt acknowledge cycle for the highest priority IP (see Section 3.3.3). In the Non-Vectored Mode (WR9 bit D5 = 1), an IUS is set during a CPU read of RR2 (channel A or B) for the highest priority IP (see Section 3.3.4). IUS is used to control the operation of the interrupt daisy chain (see Section 3.3.1) by masking lower-priority interrupts. At the end of an interrupt service routine, the processor should issue a Reset Highest IUS command in WR0 to allow lower-priority interrupt requests. This is the only way, short of a software or hardware reset, that an IUS bit may be reset.

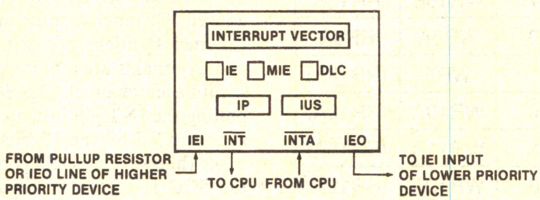
3.3.1 Daisy-Chain Priority Resolution

The six sources of interrupt in the SCC are prioritized in a fixed order via a daisy chain; provision is made, via the IEI and IEO pins, for use of an external daisy chain as well. Channel A interrupts are higher-priority than Channel B interrupts, with the receiver, transmitter, and external/status interrupts prioritized in that order within each channel. The SCC requests an interrupt by pulling the $\overline{INT}$ pin Low from its open-drain state. This is controlled by the IP bits and the IEI input, among other things. A flow-chart of the interrupt sequence for the SCC is shown in Figure 3-6. The internal daisy chain links the six sources of interrupt in a fixed order, chaining the IUS bits for each source. While an IUS is set, all lower-priority interrupt requests are masked off; during an Interrupt Acknowledge cycle the IP bits are also gated into the daisy chain. This insures that the highest-priority IP will be selected to have its IUS set.

Register	HARDWARE RESET 7	6	5	4	3	2	1	0	CHANNEL RESET 7	6	5	4	3	2	1	0
WR0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
WR1	0	0	.	0	0	.	0	0	0	0	.	0	0	.	0	0
WR2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
WR3	.	.	.	.	.	.	.	0	.	.	.	.	.	.	.	0
WR4	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.
WR5	0	.	.	0	0	0	0	.	0	.	.	0	0	0	0	.
WR6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
WR7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
WR9	1	1	0	0	0	0	.	.	.	.	0	.	.	.	.	.
WR10	0	0	0	0	0	0	0	0	0	.	.	0	0	0	0	0
WR11	0	0	0	0	1	0	0	0	.	.	.	.	.	.	.	.
WR12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
WR13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
WR14	.	.	1	0	0	0	0	0	.	.	1	0	0	0	.	.
WR15	1	1	1	1	1	0	0	0	1	1	1	1	1	0	0	0
RR0	0	1	.	.	.	1	0	0	0	1	.	.	.	1	0	0
RR1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
RR3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RR10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

230925-61

Figure 3-4. 82530 Register Reset Values



230925-12

Figure 3-5. Peripheral Interrupt Structure

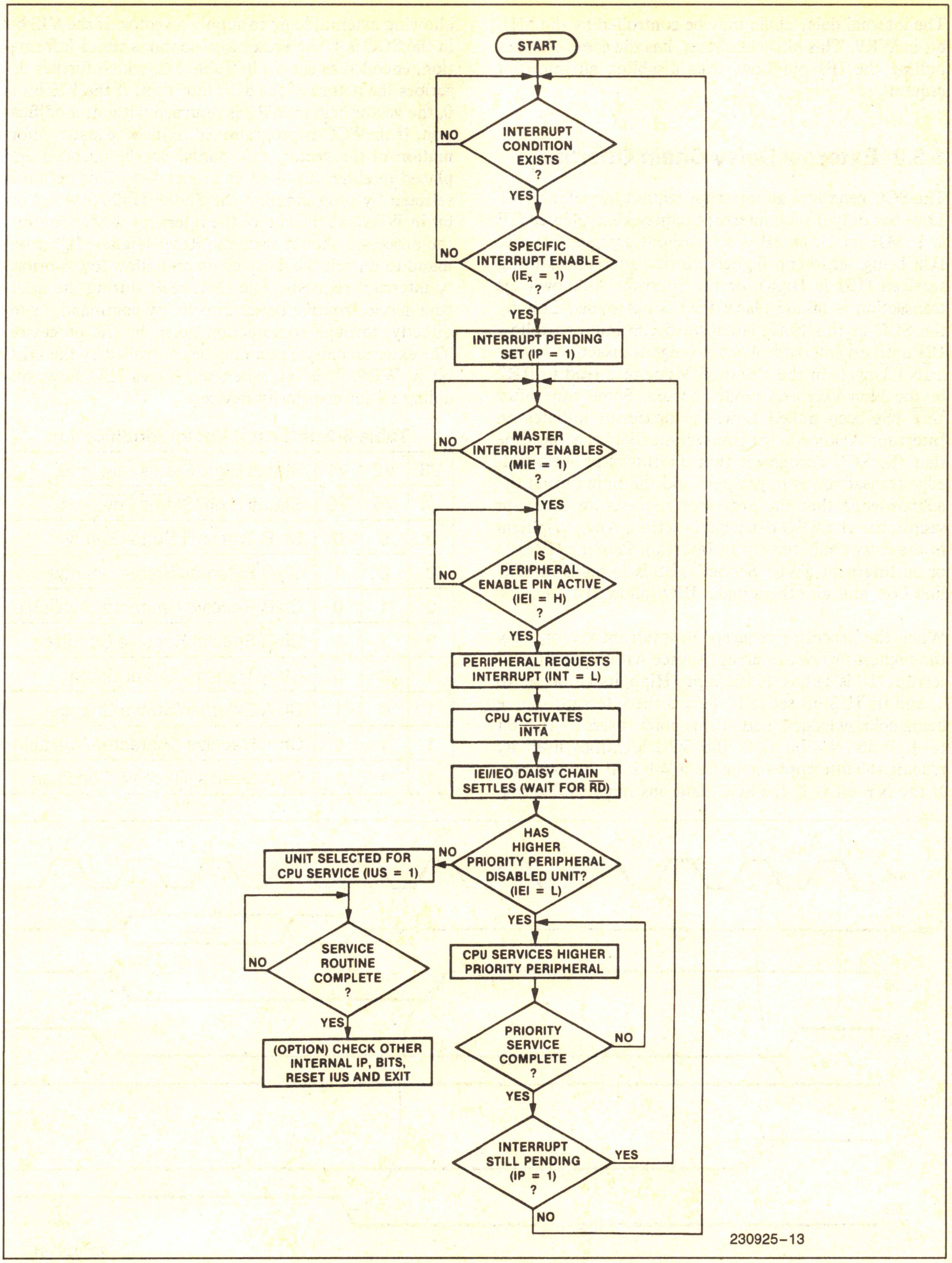


Figure 3-6. Interrupt Flowchart

The internal daisy chain may be controlled by the MIE bit in WR9. This bit, when reset, has the same effect as pulling the IEI pin Low, thus disabling all interrupt requests.

3.3.2 External Daisy Chain Operation

The SCC generates an interrupt request by pulling $\overline{INT}$ Low, but only if such interrupt requests are enabled (IE is 1, MIE is 1), an IP is set without a higher-priority IUS being set, or no higher-priority interrupt is being serviced (IEI is High), or no Interrupt Acknowledge transaction is taking place. IEO is not pulled Low by the SCC at this time, but instead continues to follow IEI until an Interrupt Acknowledge transaction (either a $\overline{INTA}$ cycle in the Vectored Mode or a read to RR2 in the Non-Vectored mode) occurs. Some time after $\overline{INT}$ has been pulled Low, the processor initiates an Interrupt Acknowledge transaction. Between the time that the SCC recognizes that an Interrupt Acknowledge transaction is in progress and the time during the acknowledge that the processor requests an interrupt vector, the IEI/IEO daisy chain settles. Any peripheral in the daisy chain having an Interrupt Pending (IP is 1) or an Interrupt Under Service (IUS is 1) holds its IEO line Low and all others make IEO follow IEI.

When the processor requests an interrupt vector, only the highest-priority interrupt source with a pending interrupt (IP is 1) has its IEI input High, its IE bit set to 1, and its IUS bit set to 0. This is the interrupt source being acknowledged, and at this point it sets its IUS bit to 1. If its NV bit is 0, the SCC identifies itself by placing the interrupt vector from WR2 on the data bus. If the NV bit is 1, the SCC data bus remains floating, allowing external logic to supply a vector. If the VIS bit in the SCC is 1, the vector also contains status information, encoded as shown in Table 3-2, which further describes the nature of the SCC interrupt. If the VIS bit is 0, the vector held in WR2 is returned without modification. If the SCC is programmed to include status information in the vector, this status may be encoded and placed in either bits 1–3 or in bits 4–6. This option is selected by programming the Status High/Status Low bit in WR9. At the end of the interrupt service routine, the processor should issue the Reset Highest IUS command to unlock the daisy chain and allow lower-priority interrupt requests. The IP is reset during the interrupt service routine either directly by command, or indirectly, through some action taken by the processor. The external daisy chain may be controlled by the DLC bit in WR9. This bit, when set, forces IEO Low, disabling all lower-priority devices.

Table 3-2. Interrupt Vector Modification

V3	V2	V1	Status High/$\overline{\text{Status Low}}$ = 0
V4	V5	V6	Status High/$\overline{\text{Status Low}}$ = 1
0	0	0	Ch B Transmit Buffer Empty
0	0	1	Ch B External/Status Change
0	1	0	Ch B Receive Character Available
0	1	1	Ch B Special Receive Condition
1	0	0	Ch A Transmit Buffer Empty
1	0	1	Ch A External/Status Change
1	1	0	Ch A Receive Character Available
1	1	1	Ch A Special Receive Condition

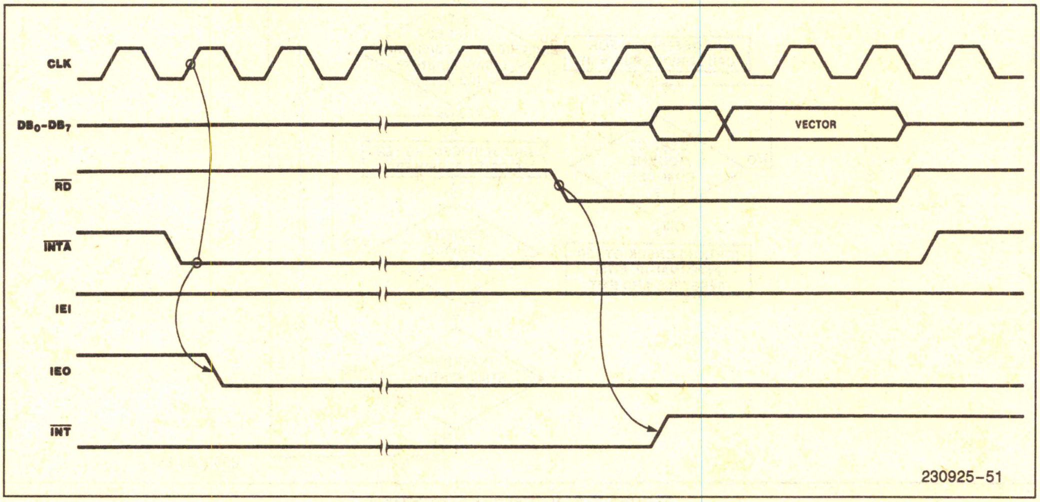


Figure 3-7. 82530 Vectored Mode Interrupt Acknowledge Details

3.3.3 82530 Vectored Mode Interrupt Acknowledge Details

The Interrupt Acknowledge cycle timing for the 82530 Vectored Mode is shown in Figure 3-7. The state of the $\overline{INTA}$ pin is latched by the rising edge of CLK. Between the time $\overline{INTA}$ is sampled Low and $\overline{RD}$ falls, the internal and external daisy chains settle. Both chains must have reached steady-state when $\overline{RD}$ falls, since it is this falling edge that sets the IUS bits in the 82530. The 82530 may be programmed to place a vector on DB_0–DB_7 while $\overline{RD}$ is low. If the falling edge of $\overline{RD}$ sets an IUS bit in the 82530, the $\overline{INT}$ pin goes inactive in response to this falling edge. The $\overline{INT}$ pin will also go inactive in response to IEI going Low during the daisy-chain settle time. Note that only one $\overline{RD}$ is allowed during an Interrupt Acknowledge cycle. Another important fact is that the IP bits in the 82530 are updated by CLK, divided by two, and this clock to update IPs is stopped while the pointers point to address two or three. This prevents changing data during a read of RR2 or RR3. Interrupt requests will thus be delayed if the pointers are left pointing at registers two or three.

3.3.4 82530 Non-Vectored Mode Interrupt Acknowledge Details*

This additional Interrupt Acknowledge mode allows the SCC user to respond to all 82530 interrupt requests with a simple software acknowledgement through RR2. Programming WR9 bit D5 to 1, enables a CPU read to RR2 to internally emulate a hardware Interrupt Acknowledge cycle as it functions in the Vectored Mode. In the Non-Vectored Mode, $\overline{INTA}$ must be externally tied inactive high and the CPU should only read the interrupt vector from RR2 once in response to each and

*NOTE:
The Non-Vectored Mode is not supported by 82530 components with copyright dates prior to 1986.

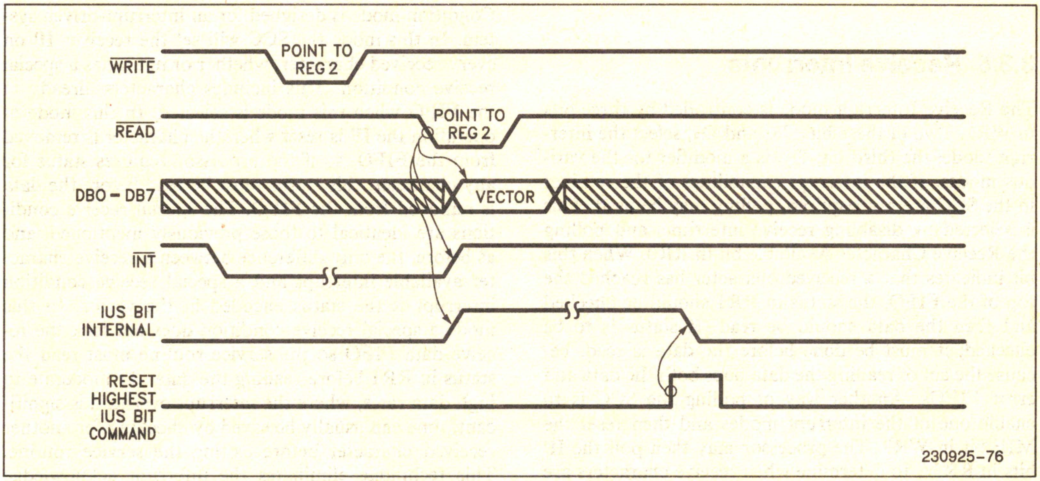


Figure 3-8. Non-Vectored Mode Interrupt Acknowledge Timing

every 82530 interrupt request. The CPU must first read RR2 to determine the internal interrupt source and then jump to the appropriate interrupt service routine. In any mode, pointing to register 2 disables any new interrupts from being accepted and will allow the internal daisy chain to reach steady state before a read or write cycle occurs. If the Non-Vectored Mode bit is enabled, a read to RR2 will set the IUS bit for the highest priority IP as shown in Figure 3-8. Since the CPU can acknowledge the SCC of highest priority with a read of its RR2 interrupt vector, there is no need for an external daisy chain in the Non-Vectored Mode. IEI for all SCC devices should be tied active high. When acknowledging a SCC interrupt request, the CPU must issue one read to RR2 per interrupt request and it should be the first CPU cycle to the 82530 during the interrupt service routine. The modified interrupt vector can be read from channel B, or the original vector stored in WR2 can be read from channel A. Either action will produce the same internal actions on the IUS logic. In the Non-Vectored Mode, the No Vector and Vector Includes Status bits in WR9 are ignored.

3.3.5 Receive Interrupts

The Receive Interrupt mode is controlled by three bits in WR1. Two of these bits, D_4 and D_3, select the interrupt mode; the third bit, D_2, is a modifier for the various modes. If the interrupt capabilities of the receiver in the SCC are not required, polling may be used. This is selected by disabling receive interrupts and polling the Receive Character Available bit in RR0. When this bit indicates that a received character has reached the top of the FIFO, the status in RR1 should be checked and then the data should be read. If status is to be checked, it must be done before the data is read, because the act of reading the data pops both the data and error FIFOs. Another way of polling the SCC is to enable one of the interrupt modes and then reset the MIE bit in WR9. The processor may then poll the IP bits in RR3A to determine when receive characters are available.

The Receive Interrupt on First Character or Special Condition mode is designed for use with DMA transfers of the receive characters. After this mode is selected, the first character received, or the first character already stored in the FIFO, will set the receiver IP. This IP will be reset when this character is removed from the SCC. No further receive interrupts will occur until the processor issues an Enable Interrupt on Next Receive Character command in WR0 or until a special receive condition occurs. The SCC recognizes several special receive conditions. A receive overrun, where a character in the FIFO is written over, is a special receive condition, as is a framing error in Asynchronous mode or the end-of-frame condition in SDLC mode. In addition, if D_2 of WR1 is set, any character with a parity error will generate a special receive condition interrupt. The correct sequence of events when using this mode is to first select the mode and wait for the receive character available interrupt. When the interrupt occurs the processor should read the character and then enable the DMA to transfer the remaining characters. A special receive condition interrupt may occur any time after the first character is received, but is guaranteed to occur after the character having the special condition has been read. The status is not lost in this case, however, because the FIFO is locked by the special condition. In the service routine the processor should read RR1 to obtain the status, and may read the data again if necessary. The FIFO is unlocked by issuing an Error Reset command in WR0. If the special condition was End-of-Frame, the processor should now issue the Enable Interrupt on Next Receive Character command to prepare for the next frame. The first character interrupt and special condition interrupt are distinguished by the status included in the interrupt vector. In all other respects they are identical, including sharing the IP and IUS bits.

The Receive Interrupt on All Characters or Special Condition mode is designed for an interrupt-driven system. In this mode the SCC will set the receiver IP on every received character, whether or not it has a special receive condition. This includes characters already in the FIFO when this mode is selected. In this mode of operation the IP is reset when the character is removed from the FIFO, so if the processor requires status for any character, this status must be read before the data is removed from the FIFO. The special receive conditions are identical to those previously mentioned, and as before, the only difference between a receive character available interrupt and a special receive condition interrupt is the status encoded in the vector. In this mode a special receive condition does not lock the receive data FIFO so the service routine must read the status in RR1 before reading the data. At moderate to high data rates, where the interrupt overhead is significant, time can usually be saved by checking for another received character before exiting the service routine. This technique eliminates the interrupt acknowledge and processor state saving time, but care must be exercised because this receive character must be checked for special receive conditions before it is removed from the SCC.

The Receive Interrupt on Special Condition Only mode is designed for use with DMA transfers of the receive characters. In this mode, only receive characters with special conditions will cause the receiver IP to be set. All other characters are assumed to be transferred via a DMA. No special initialization sequence is needed in this mode. Usually the DMA is initialized and enabled, and then this mode is selected in the SCC. A special receive condition interrupt may occur at any time after this mode is selected but the logic guarantees that the interrupt will not occur until after the character with the special condition has been read from the SCC. The special condition locks the FIFO so that the status will

be valid when read in the interrupt service routine, and guarantees that the DMA will not transfer any characters until the special condition has been serviced. In the service routine the processor should read RR1 to obtain the status and unlock the FIFO by issuing an Error Reset command. DMA transfer of the receive characters will then resume.

3.3.6 Transmit Interrupts

Transmit interrupts are controlled by the Transmit Interrupt Enable bit (D_1) in WR1. If the interrupt capabilities of the SCC are not required, polling may be used. This is selected by disabling the transmit interrupts and polling the Transmit Buffer Empty bit in RR0. When the Transmit Buffer Empty bit is set a character may be written to the SCC without fear of writing over previous data. Another way of polling the SCC is to enable the transmit interrupt and then reset the MIE bit in WR9. The processor may then poll the IP bits in RR3A to determine when the transmit buffer is empty. Transmit interrupts should also be disabled in the case of DMA transfer of the transmitted data.

While the transmit interrupts are enabled the SCC will set the transmit IP whenever the transmit buffer becomes empty. Note that this means that the transmit buffer must have been full before the transmit IP can be set. Thus when the transmit interrupts are first enabled the transmit IP will not be set until after the first character is written to the SCC. In synchronous modes one other condition can cause the transmit IP to be set. This occurs at the end of a transmission after CRC is sent. When the last bit of CRC has cleared the Transmit Shift register and the flag or sync character is loaded into the Transmit Shift register, the SCC will set the transmit IP. Data for the new frame or block to be transmitted may be written at this time. In this particular case the Transmit Buffer Empty bit in RR0 is not set; only the transmit IP is set. If the Transmit Buffer Empty bit is, in fact, set for the transmit interrupt that occurs immediately after CRC transmission, this indicates that data was written while CRC was being sent. This is an indication that the transmitter underflowed without the CPU being aware of it. The transmit IP is reset either by writing data to the transmit buffer or by issuing the Reset Transmit IP command in WR0. Ordinarily the response to a transmit interrupt is to write more data to the SCC; however, at the end of a frame or block of data where CRC is to be sent next, the Reset Transmit IP command should be issued in lieu of data.

3.3.7 External/Status Interrupts

There are several sources of External/Status interrupts, each of which may be individually enabled in WR15. The master enable for the external/status interrupts is located in WR1 (D_0). The individual enable bits in WR15 control whether or not latches will be present in the path from the source of interrupt to the status bit in RR0. If an individual enable bit in WR15 is set to 0 the latches are not present in the signal path and the value read in RR0 reflects the current status. An interrupt source whose individual enable in WR15 is 0 is not a source of External/Status interrupts even though the external/status Interrupt Enable bit is set. When an individual enable in WR15 is set to 1, the latch is present in the signal path. The latches for the sources of external/status interrupts are not independent. Rather, they all close at the same time as a result of a state change by one of the sources of interrupt. Thus, a read of RR0 returns the current status for any bits whose individual enable is 0 and either the current state or the latched state of the remainder of the bits. To guarantee the current status the processor should issue a Reset External/Status Interrupts command in WR0 to open the latches. The External/Status IP is set by the closing of the latches and remains set as long as they are closed. If the master enable for the external/status interrupts is not set the IP will never be set, even though the latches may be present in the signal paths and working as described. Because the latches close on the current status but give no indication of change, the processor must maintain a copy of RR0 in memory. When the SCC generates an external/status interrupt the processor should read RR0 and determine which condition changed state and take appropriate action. The copy of RR0 in memory must then be updated and the Reset External/Status Interrupt command issued. Care must be used in writing the interrupt service routine for the external/status interrupts because it is possible for more than one status condition to change state at the same time. All of the latch bits in RR0 should be compared to the copy of RR0 in memory. If none have changed and the ZC interrupt is enabled, the zero count condition caused the interrupt.

The operation of the individual enable bits in WR15 for each of the six sources of external/status interrupts is identical, but subtle differences exist in the operation of each source of interrupt. The six sources are Break/Abort, Underrun/EOM, CTS, DCD, Sync/Hunt and Zero Count. The Break/Abort, Underrun/EOM and Zero Count conditions are internal to the SCC, while Sync/Hunt may be internal or external, and CTS and CD are purely external signals. In the following discussions each source is assumed to be enabled, so that the latches are present, and the external/status interrupts are enabled as a whole. Recall that the External/Status IP is set while the latches are closed and that the state of the signal is reflected immediately in RR0 if the latches are not present.

The Break/Abort status is used in asynchronous and SDLC modes but is always 0 in synchronous modes other than SDLC. In asynchronous modes this bit is set when a break sequence (null character plus framing er-

ror) is detected in the receive data stream, and remains set as long as 0s continue to be received. This bit is reset when a 1 is received. A single null character is left in the receive FIFO each time that the break condition is terminated. This character should be read and discarded. In SDLC mode this bit is set by the detection of an abort sequence, which is seven or more continguous 1s in the receive data stream. The bit is reset when a 0 is received. A received abort forces the receiver into Hunt, which is also an external/status condition. Though these two bits change state at roughly the same time, one or two external/status interrupts may be generated as a result. The Break/Abort bit is unique in that both transitions are guaranteed to cause the latches to close, even if another external/status interrupt is pending at the time these transitions occur. This guarantees that a break or abort will be caught.

The Transmit Underrun/EOM bit is used in synchronous modes to control the transmission of CRC. This bit is reset by issuing the Reset Transmit Underrun/EOM command in WR0. However, this transition does not cause the latches to close; this occurs only when the bit is set. To inform the processor of this fact, the SCC sets this bit when CRC is loaded into the Transmit Shift register. This bit will also be set if the processor issues the Send Abort command in WR0. The bit is always set in Asynchronous mode.

The CTS bit reports the state of the $\overline{CTS}$ input and the CD bit reports the state of the $\overline{CD}$ input. Both bits latch on either input transition. In both cases, if the latches are closed any odd number of transitions on an input will close the latches again after the Reset External/Status interrupt command is issued, while an even number of transitions will not.

The Zero Count bit is set when the counter in the baud rate generator reaches a count of 0 and is reset when the counter is reloaded. The latches are closed only when this bit is set to 1, and the status in RR0 always reflects the current status. While the Zero Count IE bit in WR15 is reset this bit is forced to 0.

There are a variety of ways in which the Sync/Hunt may be set and reset, depending on the SCC's mode of operation. In the Asynchronous mode this bit reports the state of the $\overline{SYNC}$ pin, latching on both input transitions. The same is true of External Sync mode. However, if the crystal oscillator is enabled while in Asynchronous mode this bit will be forced to 0 and the latches will not be closed. Selecting the crystal oscillator option in External Sync mode is illegal, but the result will be the same. In Synchronous modes other than SDLC the Sync/Hunt reports the Hunt state of the receiver. Hunt mode is entered when the processor issues the Enter Hunt command in WR3. This forces the receiver to search for a sync character match in the receive data stream. Because both transitions of the Hunt bit close the latches, issuing this command will cause an external/status interrupt. The SCC resets this bit when character synchronization has been achieved, causing the latches to again be closed. In these synchronous modes the SCC will not reenter the Hunt mode automatically; only the Enter Hunt command will set this bit. In SDLC mode this bit is also set by the Enter Hunt command but the receiver will also automatically enter the Hunt mode if an Abort sequence is received. The receiver leaves Hunt upon receipt of a flag sequence. Both transitions of the Hunt bit will cause the latches to be closed. In SDLC mode the receiver will automatically synchronize on Flag characters. The receiver is in Hunt mode when it is enabled, so the Enter Hunt command will probably never be needed.

If careful attention is paid to details the interrupt service routine for external/status interrupts is straightforward. To determine which bit or bits changed state, the routine should first read RR0 and compare it to a copy from memory. For each changed bit the appropriate action should be taken and the copy in memory updated. The service routine should close with a Reset External/Status interrupts command to reopen the latches. The copy of RR0 in memory should always have the Zero Count bit set to 0, since this will be the state of the bit after the Reset External/Status interrupts command at the end of the service routine. When the processor issues the Reset Transmit Underrun/EOM latch command in WR0, the Transmit Underrun/EOM bit in the copy of RR0 in memory should be reset because this transition does not cause an interrupt.

3.4 BLOCK TRANSFER

The SCC offers several alternatives for the block transfer of data. The various options are selected in WR1 (bits D_7 through D_5) and WR14 (bit D_2). Each channel in the SCC has two pins which may be used to control the block transfer of data. Both pins in each channel may be programmed to act as DMA Request signals, and one pin in each channel may be programmed to act as a $\overline{READY}$ signal for the CPU. In either mode, it is advisable to select and enable the mode in two separate accesses of the appropriate register. The first access should select the mode and the second access should enable the function. This procedure prevents glitches on the output pins. Reset forces $\overline{READY}$ mode, with $RDY/\overline{REQ}$ open-drain.

3.4.1 $\overline{READY}$ on Transmit

The $\overline{READY}$ function on transmit is selected by setting both D_6 and D_5 of WR1 to 0 and then enabling the function by setting D_7 of WR1 to 1. In this mode the $RDY/\overline{REQ}$ pin carries the $\overline{READY}$ signal, and is open-drain when inactive and Low when active. When the processor attempts to write to the transmit buffer when it is full, the SCC will assert $\overline{READY}$ until the

buffer is empty. This allows the use of a block-move instruction to transfer the transmit data. In all other cases $\overline{READY}$ will remain open-drain. $\overline{READY}$ will go active in response to $\overline{WR}$ going active, but only if the data buffer is being accessed, either directly or via the pointers. The $\overline{READY}$ pin is released in response to the falling edge of CLK. Details of the timing are shown in Figure 3-9.

3.4.2 $\overline{READY}$ on Receive

The $\overline{READY}$ function on receive is selected by setting D_6 of WR1 to 0, D_5 of WR1 to 1, and then enabling the function by setting D_7 of WR1 to 1. In this mode the $\overline{RDY}/\overline{REQ}$ pin carries the $\overline{READY}$ signal, and is open-drain when inactive and Low when active. When the processor attempts to read data from the receive FIFO when it is empty, the SCC will assert $\overline{READY}$ until a character has reached the top of the FIFO. This allows the use of a block-move instruction to transfer the receive data. $\overline{READY}$ will go active in response to $\overline{RD}$ going active, but only if the receive data FIFO is being accessed, either directly or via the pointers. The $\overline{READY}$ pin is released in response to the falling edge of CLK. Details of the timing are shown in Figure 3-10.

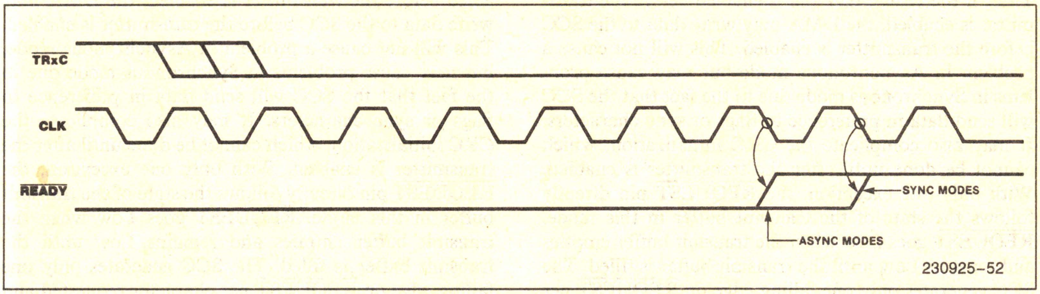


Figure 3-9. $\overline{READY}$ on Transmit

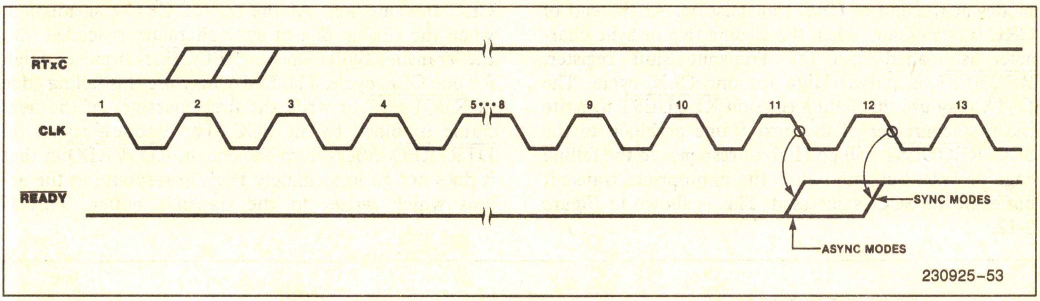


Figure 3-10. $\overline{READY}$ on Receive

3.4.3 DMA Request on Transmit (using $\overline{\text{RDY}}/\overline{\text{REQ}}$)

The Request on Transmit function is selected by setting D_6 of WR1 to 1, D_5 of WR1 to 0, and then enabling the function by setting D_7 of WR1 to 1. In this mode the $\overline{\text{RDY}}/\overline{\text{REQ}}$ pin carries the $\overline{\text{REQUEST}}$ signal, which is active Low. When this mode is selected, but not yet enabled, the $\overline{\text{RDY}}/\overline{\text{REQ}}$ pin is driven High. When the enable bit is set, $\overline{\text{REQUEST}}$ goes Low if the transmit buffer is empty at the time, or will remain High until the transmit buffer becomes empty. Note that the $\overline{\text{REQUEST}}$ pin will follow the state of the transmit buffer even though the transmitter is disabled. Thus, if the $\overline{\text{REQUEST}}$ pin is enabled before the transmitter is enabled, the DMA may write data to the SCC before the transmitter is enabled. This will not cause a problem in Asynchronous mode but may cause problems in Synchronous mode due to the fact that the SCC will send data in preference to flags or sync characters. It may also complicate the CRC initialization, which cannot be done until after the transmitter is enabled. With only one exception, the $\overline{\text{REQUEST}}$ pin directly follows the state of the transmit buffer in this mode. $\overline{\text{REQUEST}}$ goes Low when the transmit buffer empties and remains Low until the transmit buffer is filled. The SCC generates only one falling edge on $\overline{\text{REQUEST}}$ per character requested and the timing for this is shown in Figure 3-11. The one exception occurs in synchronous modes at the end of CRC transmission. At the end of CRC transmission, when the closing flag or sync character is loaded into the Transmit Shift register, $\overline{\text{REQUEST}}$ is pulsed High for one CLK cycle. The DMA may use this falling edge on $\overline{\text{REQUEST}}$ to write the first character of the next frame or block to the SCC. $\overline{\text{REQUEST}}$ will go High in response to the falling edge of $\overline{\text{WR}}$, but only when the appropriate transmit buffer in the SCC is accessed. This is shown in Figure 3-12.

3.4.4 DMA Request on Transmit (using $\overline{\text{DTR}}/\overline{\text{REQ}}$)

A second Request on Transmit function is available on the $\overline{\text{DTR}}/\overline{\text{REQ}}$ pin. This mode is selected by setting D_2 of WR14 to 1. When this bit is set to 1, $\overline{\text{REQUEST}}$ goes Low if the transmit buffer is empty at the time, or will go High until the transmit buffer becomes empty. While D_2 of WR14 is set to 0, the $\overline{\text{DTR}}/\overline{\text{REQ}}$ pin is $\overline{\text{DTR}}$ and follows the inverted state of D_7 in WR5. This pin will be High after a channel or hardware reset and in the DTR mode. In the Request mode $\overline{\text{REQUEST}}$ will follow the state of the transmit buffer even though the transmitter is disabled. Thus if $\overline{\text{REQUEST}}$ is enabled before the transmitter is enabled, the DMA may write data to the SCC before the transmitter is enabled. This will not cause a problem in Asynchronous mode, but may cause problems in Synchronous mode due to the fact that the SCC will send data in preference to flags or sync characters. It may also complicate the CRC initialization, which cannot be done until after the transmitter is enabled. With only one exception, the $\overline{\text{REQUEST}}$ pin directly follows the state of the transmit buffer in this mode. $\overline{\text{REQUEST}}$ goes Low when the transmit buffer empties and remains Low until the transmit buffer is filled. The SCC generates only one falling edge on $\overline{\text{REQUEST}}$ per character requested and the timing for this is shown in Figure 3-11. The one exception occurs in synchronous modes at the end of CRC transmission. At the end of CRC transmission, when the closing flag or sync character is loaded into the Transmit Shift register, $\overline{\text{REQUEST}}$ is pulsed High for one CLK cycle. The DMA may use this falling edge of $\overline{\text{REQUEST}}$ to write the first character of the next frame or block to the SCC. The Request signal on $\overline{\text{DTR}}/\overline{\text{REQ}}$ differs from the one on $\overline{\text{RDY}}/\overline{\text{REQ}}$ in that it does not go immediately High in response to the access which writes to the transmit buffer. This is

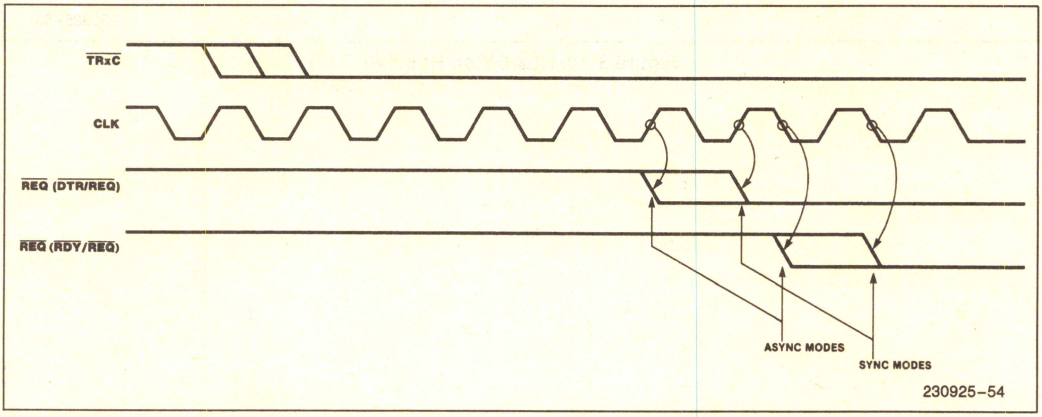


Figure 3-11. Transmit Request Assertion

because the registers in the SCC are not written during the actual access, but are delayed by some number of CLK cycles. The Request signal on $\overline{\text{DTR}}/\overline{\text{REQ}}$ follows the state of the transmit buffer exactly while the Request signal on $\overline{\text{RDY}}/\overline{\text{REQ}}$ goes inactive in anticipation of the transmit buffer becoming full. The timing of the Request signal is shown in Figure 3-12.

3.4.5 DMA Request on Receive

The Request on Receive function is selected by setting D_6 and D_5 of WR1 to 1 and then enabling the function by setting D_7 of WR1 to 1. In this mode the $\overline{\text{RDY}}/\overline{\text{REQ}}$ pin carries the $\overline{\text{REQUEST}}$ signal, which is active Low. When this mode is selected, but not yet enabled, the $\overline{\text{RDY}}/\overline{\text{REQ}}$ pin is driven High. When the enable bit is set $\overline{\text{REQUEST}}$ goes Low if the receive buffer contains a character at the time, or will remain High until a character enters the receive buffer. Note that the $\overline{\text{REQUEST}}$ pin will follow the state of the receive buffer even though the receiver is disabled. Thus if the receiver is disabled and $\overline{\text{REQUEST}}$ is still enabled the DMA will transfer the previously received data correctly. In this mode the $\overline{\text{REQUEST}}$ pin directly follows the state of the receive buffer with only one exception. $\overline{\text{REQUEST}}$ goes Low when a character enters the receive buffer and remains Low until this character is removed from the receive buffer. The SCC generates only one falling edge on $\overline{\text{REQUEST}}$ per character transfer requested and the timing for this is shown in Figure 3-13. The one exception occurs in the case of a special receive condition in the Receive Interrupt First Character or Special condition mode, or the Receive Interrupt on Special Condition Only mode. In these two interrupt modes any receive character with a special receive condition is locked at the top of the FIFO until an Error Reset command is issued. This character in the receive FIFO would ordinarily cause additional DMA Requests after the first time it is read. However, the logic in the SCC guarantees only one falling edge on $\overline{\text{REQUEST}}$ by holding $\overline{\text{REQUEST}}$ High from the time the character with the special receive condition is read, and the FIFO locked, until after the Error Reset command has been issued. Once the FIFO is unlocked by the Error Reset command, $\overline{\text{REQUEST}}$ again follows the state of the receive buffer. $\overline{\text{REQUEST}}$ will go High in response to the falling edge of $\overline{\text{RD}}$, but only when the appropriate receive buffer in the SCC is accessed. This is shown in Figure 3-14.

3.5 ASYNCHRONOUS MODE

The SCC supports Asynchronous mode with a number of programmable options including the number of bits

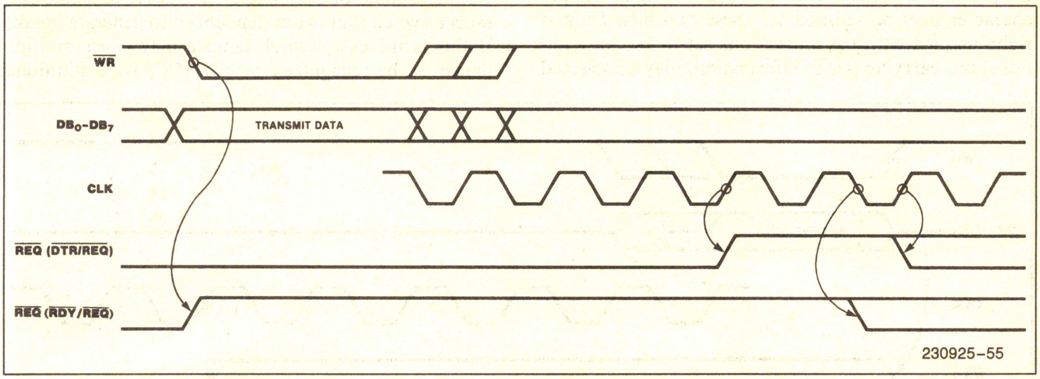


Figure 3-12. 82530 Transmit Request Release

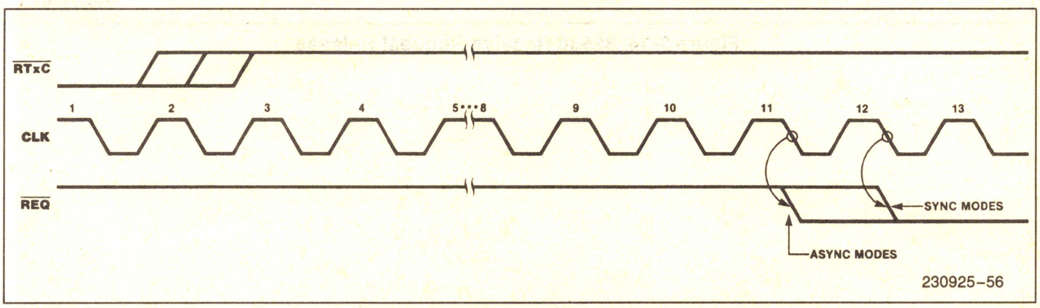


Figure 3-13. Receive Request Assertion

per character, the number of stop bits, the clock factor, modem interface signals and break detect and generation. Asynchronous mode is selected by programming the desired number of stop bits in D_3 and D_2 of WR4. Programming these two bits with other than 00 places both the receiver and transmitter in Asynchronous mode. In this mode the state of bits D_4, D_3, D_2 and D_1 of WR3, bits D_5 and D_4 of WR4, bits D_2 and D_0 of WR5, all of WR6 and WR7 and all of WR10 except D_6 and D_5, are ignored by the SCC. Bits that are ignored may be programmed with 1 or 0 or not at all.

3.5.1 Asynchronous Receive

Asynchronous mode is selected by specifying the number of stop bits per character in WR4. This selection applies only to the transmitter, however, as the receiver always checks for one stop bit. If after character assembly the receiver finds this stop bit to be a 0, the Framing Error bit in the receive error FIFO is set at the same time that the character is transferred to the receive data FIFO. This error bit accompanies the data to the top of the FIFO, where it is a special receive condition. The Framing Error bit is not latched, and so must be read in RR1 before the accompanying data is read.

The number of bits per character is controlled by bits D_7 and D_6 of WR3. Five, six, seven or eight bits per character may be selected via these two bits. Data is right-justified with the unused bits set to 1s. An additional bit, carrying parity information, may be selected by setting bit D_0 of WR4 to 1. Note that this also enables parity for the transmitter. The parity sense is selected by bit D_1 of WR4. If this bit is set to 1, the received character is checked for even parity and, if set to 0, the received character is checked for odd parity. The additional bit per character that is parity is transferred to the receive data FIFO along with the data if the data plus parity is eight bits or less. The Parity Error bit in the receive error FIFO may be programmed to cause a special receive condition interrupt by setting bit D_2 of WR1 to 1. This error bit is latched and so will remain active, once set, until an Error Reset command has been issued. If interrupts are not used to transfer data the Parity Error, Framing Error, and Overrun Error bits in RR1 should be checked before the data is removed from the receive data FIFO.

The break condition is continuous 0s, as opposed to the usual continuous ones during an idle. The SCC recognizes the Break condition upon seeing a null character (all 0s) plus a framing error. Upon recognizing this sequence the Break bit in RR0 will be set and will remain set until a 1 is received. At this point the break condition is no longer present. At the termination of a break the receive data FIFO contains a single null character, which should be read and discarded. The Framing Error bit will not be set for this character, but if odd parity has been selected, the Parity Error bit will be set. Caution should be exercised if the receive data line contains a switch that is not debounced to generate breaks. If this is the case, switch bounce may cause multiple breaks to be recognized by the SCC, with additional

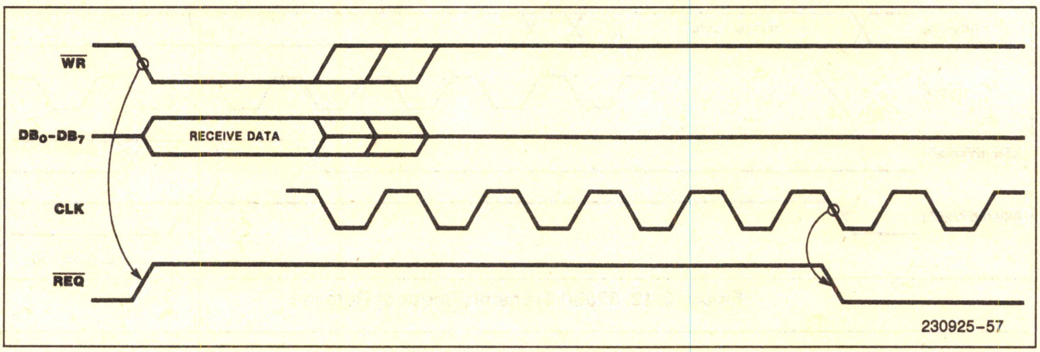


Figure 3-14. 82530 Receive Request Release

characters assembled in the receive data FIFO as well as the possibility of a receive overrun condition being latched.

The SCC may be programmed to accept a receive clock that is one, sixteen, thirty-two, or sixty-four times the data rate. This is selected by bits D_7 and D_6 in WR4. The 1X mode is used when bit synchronization external to the receiver is present. The 1X mode is the only mode in which a data encoding method other than NRZ may be used. The clock factor is common to the receiver and transmitter.

The SCC provides up to three modem control signals associated with the receiver. The $\overline{SYNC}$ pin is a general-purpose input whose state is reported in the Sync/Hunt bit in RR0. If the crystal oscillator is enabled this pin is not available and the Sync/Hunt bit is forced to 0. Otherwise, the $\overline{SYNC}$ pin may be used to carry the Ring Indicator signal. The $\overline{DTR/REQ}$ pin carries the inverted state of the DTR bit (D_7) in WR5 unless this pin has been programmed to carry a DMA Request signal. The $\overline{CD}$ pin is ordinarily a simple input to the CD bit in RR0. However, if the Auto Enables mode is selected by setting D_5 of WR3 to 1, this pin becomes an enable for the receiver. That is, if Auto Enables is on and the $\overline{CD}$ pin is High, the receiver is disabled; while the $\overline{CD}$ pin is Low, the receiver is enabled.

The initialization sequence for the receiver in Asynchronous mode is WR4 first to select the mode, then WR3 and WR5 to select the various options. At this point, the other registers should be initialized as necessary. When all of this is complete the receiver may be enabled by setting bit D_0 of WR3 to 1.

3.5.2 Asynchronous Transmit

Asynchronous mode is selected by specifying the number of stop bits per character in bits D_3 and D_2 of WR4. The three options available are one, one-and-a-half, or two stop bits per character. These two bits only select the number of stop bits for the transmitter, as the receiver always checks for one stop bit.

The number of bits per transmitted character is controlled both by bits D_6 and D_5 in WR5 and the way the data is formatted within the transmit buffer. The bits in WR5 allow the option of five, six, seven, or eight bits per character. When five bits per character is selected the data may be formatted before being written to the transmit buffer to allow transmission of from one to five bits per character. This formatting is shown in Table 3-3. In all cases the data must be right-justified, with the unused bits being ignored except in the case of five bits per character. An additional bit, carrying parity information, may be automatically appended to every transmitted character by setting bit D_0 of WR4 to 1. This bit is sent in addition to the number of bits specified in WR4 or by the data format. The parity sense is selected by bit D1 of WR4. If this bit is set to 1, the transmitter sends even parity and, if set to 0, the parity is odd.

Table 3-3. Data Format—Five Bits or Less

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	One data bit
1	1	1	0	0	0	D	D	Two data bits
1	1	0	0	0	D	D	D	Three data bits
1	0	0	0	D	D	D	D	Four data bits
0	0	0	D	D	D	D	D	Five data bits

The transmitter may be programmed to send a Break by setting bit D_4 of WR5 to 1. The transmitter will send continuous 0s from the first transmit clock edge after this command is issued, until the first transmit clock edge after this bit is reset. The transmit clock edges referred to here are those that define transmitted bit cell boundaries.

An additional status bit for use in Asynchronous mode is available in bit D_0 of RR1. This bit, called All Sent, is set when the transmitter is completely empty and any previous data or stop bits have reached the TxD pin. The All Sent bit can be used by the processor as an indication that the transmitter may be safely disabled.

The SCC may be programmed to accept a transmit clock that is one, sixteen, thirty-two, or sixty-four times the data rate. This is selected by bits D_7 and D_6 and WR4, in common with the clock factor for the receiver. Note that the chosen clock factor may restrict the number of stop bits that may be transmitted. In particular, when the clock rate and data rate are identical, one-and-a-half stop bits is not allowed. If some length other than one stop bit is desired in the times one mode, only two stop bits may be used.

There are two modem control signals associated with the transmitter provided by the SCC. The $\overline{RTS}$ pin is a simple output that carries the inverted state of the RTS bit (D_1) in WR5, unless the Auto Enables bit (D_5) is set in WR3. When Auto Enables is set the $\overline{RTS}$ pin will immediately go Low when the $\overline{RTS}$ bit is set. However, when the $\overline{RTS}$ bit is reset the RTS pin remains Low until the transmitter is completely empty and the last stop bit has left the TxD pin. Thus the $\overline{RTS}$ pin may be used to disable external drivers for the transmit data. The $\overline{CTS}$ pin is ordinarily a simple input to the CTS bit in RR0. However, if Auto Enables mode is selected this pin becomes an enable for the transmitter. That is, if Auto Enables is on and the $\overline{CTS}$ pin is High, the transmitter is disabled; the transmitter is enabled while the $\overline{CTS}$ pin is Low.

The initialization sequence for the transmitter in Asynchronous mode is WR4 first to select the mode, then WR3 and WR5 to select the various options. At this point the other registers should be initialized as necessary. When all of this is complete, the transmitter may be enabled by setting bit D_3 of WR5 to 1. Note that the transmitter and receiver may be initialized at the same time.

3.6 SYNCHRONOUS MODES

In synchronous modes of operation a special bit pattern is used to provide character synchronization. The SCC offers several options to support Synchronous mode including multiple sync character lengths, the number of bits per data character, parity generation and checking, CRC generation and checking, as well as modem controls and a transmitter to receiver synchronization function (see Figure 3-15). Synchronous mode is selected by programming bits D_3 and D_2 of WR4 with 0s. This selects Synchronous mode, as opposed to Asynchronous mode, but this selection is further modified by bits D_5 and D_7 of WR4 as well as bits D_1 and D_0 of WR10. The sync character or characters are written in WR6 and WR7. In all synchronous modes, except External Sync the state of bits D_7 and D_6 of WR4 are forced to 0 to select the times one clock mode. In External Sync mode these two bits should be programmed with 0s.

3.6.1 Synchronous Receive

The receiver in the SCC searches for character synchronization only while it is in Hunt mode. In this mode the receiver is idle except that it is searching the incoming data stream for a sync character match. The receiver is in Hunt mode when it is first enabled and may be placed in Hunt mode by command from the processor. This is accomplished by issuing the Enter Hunt Mode command in WR3. This bit (D_4) is a command, so writing a 0 to it has no effect. The hunt status of the receiver is reported by the Sync/Hunt bit in RR0. Sync/Hunt is one of the possible sources of external/status interrupts, with both transitions causing an interrupt. This is true even if the Sync/Hunt bit is set as a result of the processor issuing the Enter Hunt Mode command.

An 8-bit sync character is selected by setting bits D_5 and D_4 of WR4, as well as bit D_0 of WR10, to 0. With this option the receiver searches the data stream for a match with the eight bits in WR7. The 6-bit sync option requires the same programming except that D_0 of WR10 is set to 1 and the sync character is held in the high-order six bits of WR7. The SCC also allows the option of double length sync characters. This is selected by setting bit D_5 of WR4 to 0 and bit D_4 of WR4 to 1. The selection between 12 and 16 bits of sync character is controlled by bit D_0 of WR10. A 0 selects 16 bits of sync character, while a 1 in this bit selects a 12-bit sync character. The arrangement of the sync character in WR6 and WR7 is shown in Figure 3-16. For those applications requiring some other sync character length, the SCC makes provision for an external circuit to provide a character synchronization signal on the $\overline{\text{SYNC}}$ pin. This mode is selected by setting bits D_5 and D_4 of WR4 to 1. In this mode the Sync/Hunt bit in RR0 reports the state of the $\overline{\text{SYNC}}$ pin but the receiver must still be placed in Hunt mode when the external logic is searching for a sync character match. When the receiver is in Hunt mode and the $\overline{\text{SYNC}}$ pin is driven Low, two receive clock cycles after the last bit of the sync character is received, character assembly will begin on the rising edge of the receive clock immediately preceding the activation of $\overline{\text{SYNC}}$. This is shown in Figure 3-17. The receiver leaves Hunt mode when $\overline{\text{SYNC}}$ is driven Low. In all cases except External Sync mode the $\overline{\text{SYNC}}$ pin is an output that is driven Low by the SCC to signal that a sync character has been received. The $\overline{\text{SYNC}}$ pin is activated regardless of character boundaries so any external circuitry using it should only respond to the $\overline{\text{SYNC}}$ pulse that occurs while the receiver is in Hunt mode. The timing for the $\overline{\text{SYNC}}$ signal is shown in Figure 3-18.

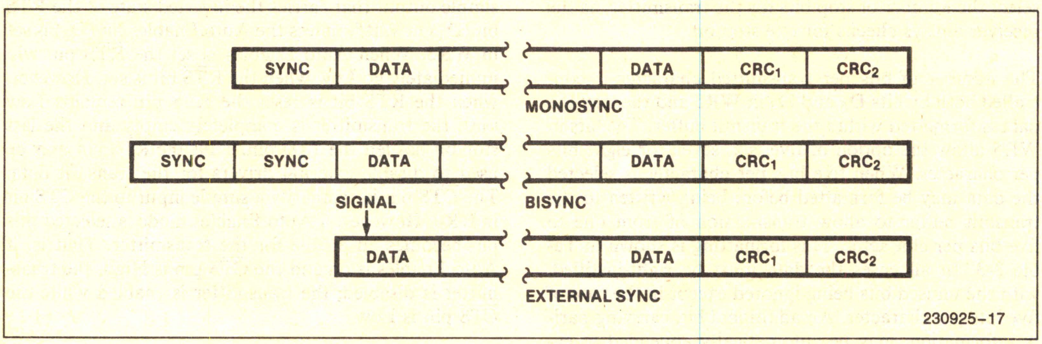


Figure 3-15. 82530 Synchronous Protocols

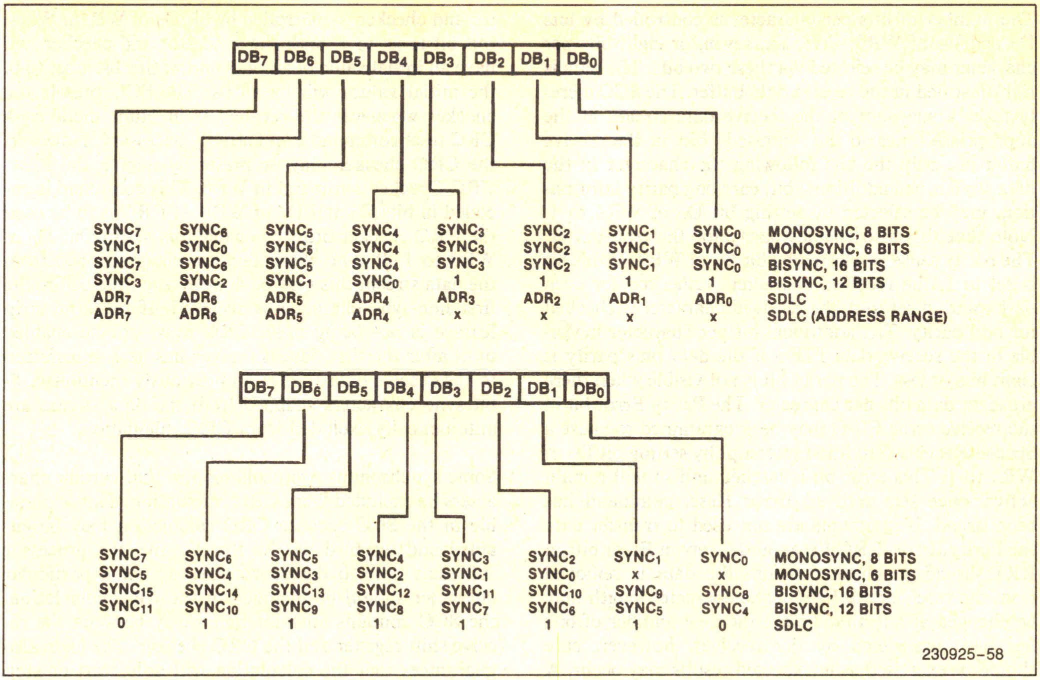


Figure 3-16. Sync Character Programming

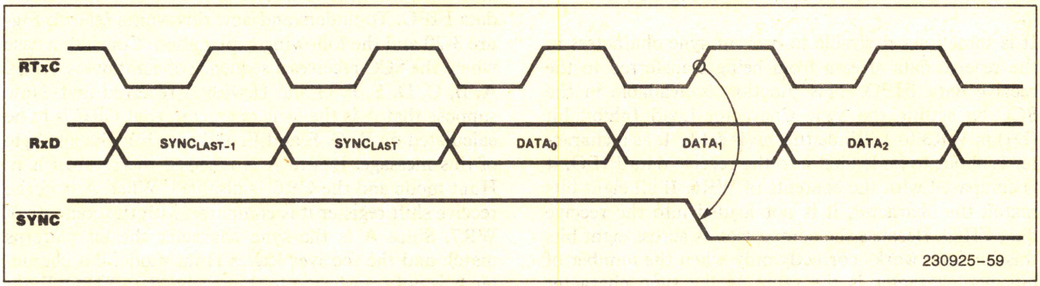


Figure 3-17. SYNC as an Input

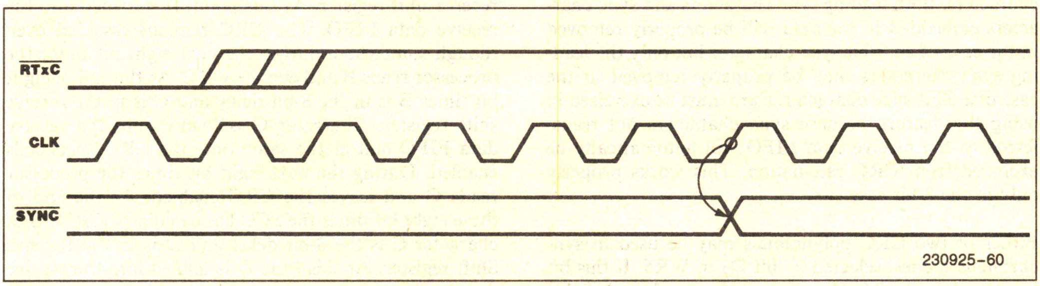


Figure 3-18. SYNC as an Output

The number of bits per character is controlled by bits D_7 and D_6 of WR3. Five, six, seven, or eight bits per character may be selected via these two bits. The data is right-justified in the receive data buffer. The SCC merely takes a snapshot of the receive data stream at the appropriate times so the "'unused" bits in the receive buffer are only the bits following the character in the data stream. An additional bit, carrying parity information, may be selected by setting bit D_0 of WR4 to 1. Note that this also enables parity for the transmitter. The parity sense is selected by bit D_1 of WR4. If this bit is set to 1, the received character is checked for even parity and, if set to 0, the received character is checked for odd parity. The additional bit per character is visible in the receive data FIFO if the data plus parity is eight bits or less. The parity bit is not visible when there are eight data bits per character. The Parity Error bit in the receive error FIFO may be programmed to cause a Special Receive Condition interrupt by setting bit D_2 of WR1 to 1. This error bit is latched and so will remain active, once set, until an Error Reset command has been issued. If interrupts are not used to transfer data the Parity Error, CRC Error, and Overrun Error bits in RR1 should be checked before the data is removed from the receive data FIFO. The character length may be changed at any time before the new number of bits has been assembled by the receiver, however, care should be exercised as unexpected results may occur. A representative example, switching from five bits to eight bits and back to five bits is shown in Figure 3-19.

It is sometimes desirable to prevent sync characters in the receive data stream from being transferred to the receive data FIFO. This function is available in the SCC by setting the Sync Character Load Inhibit bit (D_1) in WR3 to 1. While this bit is set to 1, as a character is about to be loaded into the receive data FIFO, it is compared with the contents of WR6. If all eight bits match the character, it is not loaded into the receive data FIFO. Because the comparison is across eight bits this function works correctly only when the number of bits per character is the same as the sync character length. Thus it cannot be used with 12- or 16-bit sync characters. Both leading sync characters and sync characters embedded in the data will be properly removed in the case of an 8-bit sync character but only the leading sync characters may be properly removed in the case of a 6-bit sync character. Care must be exercised in using this feature because sync characters not transferred to the receive data FIFO will automatically be excluded from CRC calculation. This works properly only in the 8-bit case.

Either of two CRC polynomials may be used in synchronous modes, selected by bit D_2 in WR5. If this bit is set to 1, the CRC-16 polynomial is used, and if this bit is set to 0, the CRC-CCITT polynomial is used. This bit controls the polynomial selection for both the receiver and transmitter. The initial state of the generator and checker is controlled by bit D_7 of WR10. When this bit is set to 1, both the generator and checker will have an initial value of all ones and, if this bit is set to 0, the initial values will be all 0s. The SCC presets the checker whenever the receiver is in Hunt mode so a CRC reset command is not strictly necessary. However, the CRC checker may be preset by issuing the Reset CRC Checker command in WR0. This command is encoded in bits D_7 and D_6 of WR0. If CRC is to be used the CRC checker must be enabled by setting bit D_0 of WR3 to 1. If sync characters are being stripped from the data stream this may be done at any time before the first non-sync character is received. If the sync strip feature is not being used, CRC must not be enabled until after the first data character has been transferred to the receive data FIFO. As previously mentioned, 8-bit sync characters stripped from the data stream are automatically excluded from CRC calculation.

Some synchronous protocols require that certain characters be excluded from CRC calculation. This is possible in the SCC because CRC calculation may be enabled and disabled on the fly. To give the processor sufficient time to decide whether or not a particular character should be included in the CRC calculation, the SCC contains an 8-bit time delay between the receive shift register and the CRC checker. The logic also guarantees that the calculation will only start or stop on a character boundary by delaying the enable or disable until the next character is loaded into the receive data FIFO. To understand how this works refer to Figure 3-20 and the following explanation. Consider a case where the SCC receives a sequence of eight bytes, called A, B, C, D, E, F, G and H with A received first. Now suppose that A is the sync character, that CRC is to be calculated on B, C, E and F, and that F is the last byte of this message. Before A is received the receiver is in Hunt mode and the CRC is disabled. When A is in the receive shift register it is compared with the contents of WR7. Since A is the sync character the bit patterns match and the receiver leaves Hunt mode, but character A is not transferred to the receive data FIFO. Eight bit times later A is in the 8-bit delay and B is in the receive shift register. At this point, B is loaded into the receive data FIFO. The CRC remains disabled even though somewhere during the next eight bit times the processor reads B and enables CRC. At the end of eight bit times B is in the 8-bit delay and C is in the receive shift register. Character C is loaded into the receive data FIFO and at the same time the CRC checker is enabled. During the next eight bit times the processor reads C and leaves the CRC enabled. At the end of these eight bit times the SCC has calculated CRC on B, character C is the 8-bit delay and D is in the Receive Shift register. At this time D is loaded into the receive data buffer and at some point during the next eight bit times the processor reads D and disables CRC. At the end of these eight bit times CRC has been calculated on C, character D is in the 8-bit delay and E is in the Receive Shift register.

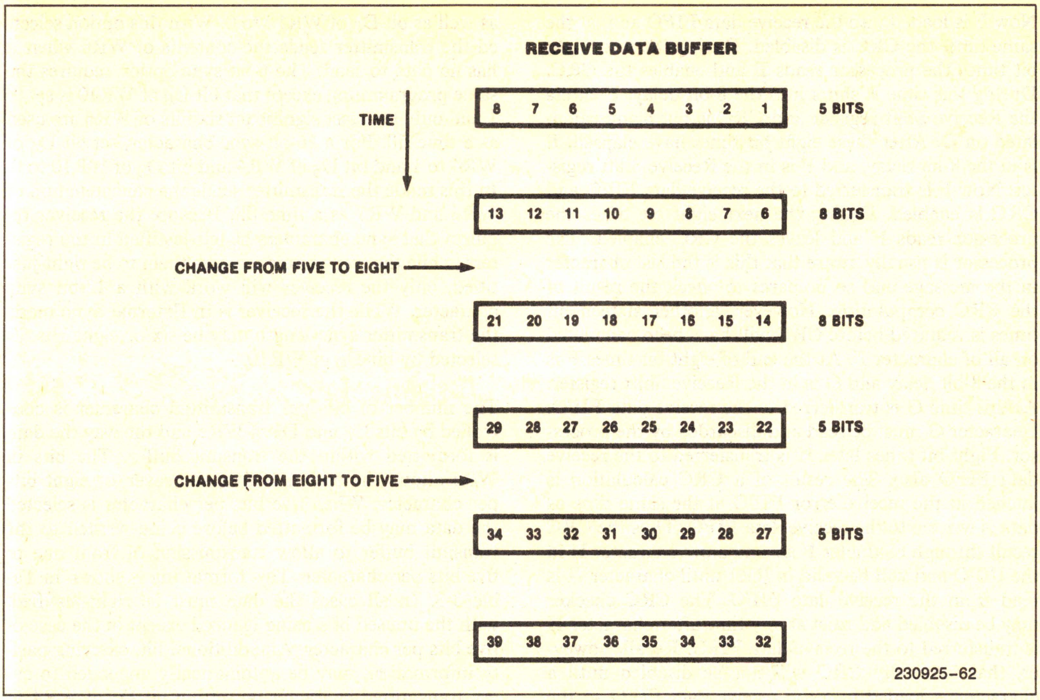


Figure 3-19. Changing Character Length

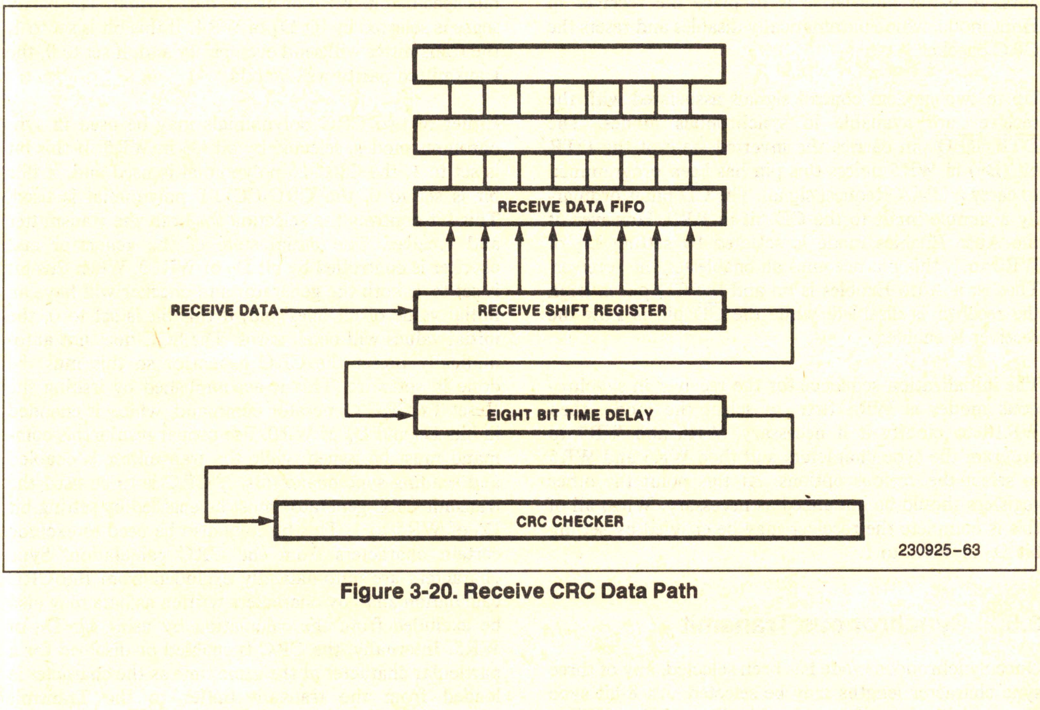


Figure 3-20. Receive CRC Data Path

Now E is loaded into the receive data FIFO and, at the same time, the CRC is disabled. During the next eight bit times the processor reads E and enables the CRC. During this time E shifts into the 8-bit delay, F enters the Receive Shift register and CRC is not being calculated on D. After these eight bit times have elapsed, E is in the 8-bit delay, and F is in the Receive Shift register. Now F is transferred to the receive data FIFO and CRC is enabled. During the next eight bit times the processor reads F and leaves the CRC enabled. The processor is usually aware that this is the last character in the message and so prepares to check the result of the CRC computation. However, another sixteen bit times is required before CRC will have been calculated on all of character F. At the end of eight bit times F is in the 8-bit delay and G is in the Receive Shift register. At this time G is transferred to the receive data FIFO. Character G must be read and discarded by the processor. Eight bit times later H is transferred to the receive data FIFO also. The result of a CRC calculation is latched in the receive error FIFO at the same time as data is written to the receive data FIFO. Thus the CRC result through character F accompanies character H in the FIFO and will be valid in RR1 until character H is read from the receive data FIFO. The CRC checker may be disabled and reset at any time after character H is transferred to the receive data FIFO. Recall, however, that internally CRC will not be disabled until a character is loaded into the receive data FIFO so the reset command should not be issued until after this occurs. A better alternative is to place the receiver in Hunt mode, which automatically disables and resets the CRC checker.

Up to two modem control signals associated with the receiver are available in synchronous modes. The $\overline{DTR}/\overline{REQ}$ pin carries the inverted state of the DTR bit (D_7) in WR5 unless this pin has been programmed to carry a DMA Request signal. The $\overline{CD}$ pin is ordinarily a sample input to the CD bit in RR0. However, if the Auto Enables mode is selected by setting D_5 of WR3 to 1, this pin becomes an enable for the receiver. That is, if Auto Enables is on and the $\overline{CD}$ pin is High the receiver is disabled; while the $\overline{CD}$ pin is Low the receiver is enabled.

The initialization sequence for the receiver in synchronous modes is WR4 first, to select the mode, then WR10 to modify it if necessary, WR6 and WR7 to program the sync characters and then WR3 and WR5 to select the various options. At this point the other registers should be initialized as necessary. When all of this is complete the receiver may be enabled by setting bit D_0 of WR3 to 1.

3.6.2 Synchronous Transmit

Once Synchronous mode has been selected, any of three sync character lengths may be selected. An 8-bit sync character is selected by setting bits D_5 and D_4 of WR4, as well as bit D_0 of WR10 to 0. With this option selected the transmitter sends the contents of WR6 when it has no data to send. The 6-bit sync option requires the same programming except that bit D_0 of WR10 is set to 1 and only the least significant six bits of WR6 are used as a time fill. For a 16-bit sync character, set bit D_4 of WR4 to 1 and bit D_5 of WR4 and bit D_0 of WR10 to 0. In this mode the transmitter sends the concatenation of WR6 and WR7 as a time fill. Because the receiver requires that sync characters be left-justified in the registers, while the transmitter requires them to be right-justified, only the receiver will work with a 12-bit sync character. While the receiver is in External Sync mode the transmitter sync length may be six or eight bits, as selected by bit D_0 of WR10.

The number of bits per transmitted character is controlled by bits D_6 and D_5 of WR5 and the way the data is formatted within the transmit buffer. The bits in WR5 allows the option of five, six, seven or eight bits per character. When five bits per character is selected the data may be formatted before being written to the transmit buffer to allow transmission of from one to five bits per character. This formatting is shown in Table 3-3. In all cases the data must be right-justified, with the unused bits being ignored except in the case of five bits per character. An additional bit, carrying parity information, may be automatically appended to every transmitted character by setting bit D_0 of WR4 to 1. This parity bit is sent in addition to the number of bits specified in WR4 or by the data format. The parity sense is selected by bit D_1 of WR4. If this bit is set to 1, the transmitter will send even parity and, if set to 0, the transmitted parity will be odd.

Either of two CRC polynomials may be used in synchronous modes, selected by bit D_2 in WR5. If this bit is set to 1, the CRC-16 polynomial is used and, if this bit is set to 0, the CRC-CCITT polynomial is used. This bit controls the selection for both the transmitter and receiver. The initial state of the generator and checker is controlled by bit D_7 of WR10. When this bit is set to 1, both the generator and checker will have an initial value of all ones and, if this bit is set to 0, the initial values will be all zeros. The SCC does not automatically preset the CRC generator so this must be done in software. This is accomplished by issuing the Reset Tx CRC Generator command, which is encoded in bits D_7 and D_6 of WR0. For proper results this command must be issued while the transmitter is enabled and sending sync characters. If CRC is to be used the transmit CRC generator must be enabled by setting bit D_0 of WR5 to 1. This bit may also be used to exclude certain characters from the CRC calculation. Sync characters are automatically excluded from the CRC calculation and any characters written as data may also be excluded from the calculation by using bit D_0 of WR5. Internally, the CRC is enabled or disabled for a particular character at the same time as the character is loaded from the transmit buffer to the Transmit

Shift register. Thus, to exclude a character from CRC calculation bit, D_0 of WR5 should be set to 0 before the character is written to the transmit buffer. This guarantees that the internal disable will occur when the character moves from the buffer to the shift register. Once the buffer becomes empty, the Tx CRC Enable bit may be written for the next character.

Enabling the CRC generator is not sufficient to control the transmission of CRC. In the SCC this function is controlled by the Tx Underrun/EOM bit, which may be reset by the processor and set by the SCC. When the transmitter underruns (both the transmit buffer and Transmit Shift register are empty) the state of the Tx Underrun/EOM bit determines the action taken by the SCC. If the Tx Underrun/EOM bit is set when the underrun occurs, the transmitter will send sync characters and, if this bit is reset when the underrun occurs, the transmitter will send the accumulated CRC followed by sync characters. When the CRC is loaded into the Transmit Shift register for transmission the SCC will set the Tx Underrun/EOM bit to indicate this. This transition may be programmed to cause an external/status interrupt, or the Tx Underrun/EOM bit is available in RR0. The Reset Tx Underrun/EOM Latch command is encoded in bits D_7 and D_6 of WR0. For correct transmission of the CRC at the end of a block of data, this command must be issued after the first character is written to the SCC but before the transmitter underruns after the last character written to the SCC but before the transmitter underruns after the last character written to the SCC. The command is usually issued immediately after the first character is written to the SCC so that CRC will be sent if an underrun occurs inadvertently during the block of data.

In synchronous modes, if the transmitter is disabled during transmission of a character, that character will be sent completely. This applies to both data and sync characters. However, if the transmitter is disabled during the transmission of CRC, the 16-bit transmission will be completed, but the remaining bits will come from the SYNC registers rather than the remainder of the CRC.

There are two modem control signals associated with the transmitter provided by the SCC. The $\overline{RTS}$ pin is a simple output that carries the inverted state of the RTS bit (D_1) in WR5. The $\overline{CTS}$ pin is ordinarily a simple input to the CTS bit in RR0. However, if Auto Enables mode is selected this pin becomes an enable for the transmitter. That is, if Auto Enables is on and the $\overline{CTS}$ pin is High the transmitter is disabled, while the transmitter is enabled while the $\overline{CTS}$ pin is Low.

The initialization sequence for the transmitter in synchronous modes is WR4 first, to select the mode, then WR10 to modify it if necessary, WR6 and WR7 to program the sync characters and then WR3 and WR5 to select the various options. At this point, the other registers should be initialized as necessary. When all of this is complete the transmitter may be enabled by setting bit D_3 of WR5 to 1. Now that the transmitter is enabled the CRC generator may be initialized by issuing the Reset Tx CRC Generator command in WR0.

3.6.3 Transmitter to Receiver Synchronization

The SCC contains a transmitter to receiver synchronization functions that may be used to guarantee that the character boundaries for the received and transmitted data are the same. In this mode the receiver is in Hunt and the transmitter is idle, sending either all 1s or all 0s. When the receiver recognizes a sync character it leaves Hunt mode and one character time later the transmitter is enabled and begins sending sync characters. Beyond this point the receiver and transmitter are again completely independent, except that the character boundaries are now aligned. This is shown in Figure 3-21. There are several restrictions on the use of this feature in the SCC. First, it will only work with 6-bit, 8-bit or 16-bit sync characters, and the data character length for both the receiver and the transmitter must be six bits with a 6-bit sync character and eight bits with an 8-bit or 16-bit sync character. Of course, the receive and transmit clocks must have the same rate as well as the proper phase relationship.

A specific sequence of operations must be followed to synchronize the transmitter to the receiver. Both the receiver and transmitter must have been initialized for operation in Synchronous mode sometime in the past, although this initialization need not be redone each time the transmitter is synchronized to the receiver. The transmitter is disabled by setting bit D_3 of WR5 to 0. At this point the transmitter will send continuous 1s. If it is desired that continuous 0s be transmitted, the Send Break bit (D_4) in WR5 should be set to 1. The transmitter is now idling but must still be placed in the transmitter to receiver synchronization mode. This is accomplished by setting the Loop Mode bit (D_1) in WR10 and then enabling the transmitter by setting bit D_3 of WR5 to 1. At this point the processor should set the Go Active on Poll bit (D_4) in WR10. The final step is to force the receiver to search for sync characters. If the receiver is currently disabled the receiver will enter Hunt mode when it is enabled by setting bit D_0 of WR3 to 1. If the receiver is already enabled it may be placed in Hunt mode by setting bit D_4 of WR3 to 1. Once the receiver leaves Hunt mode the transmitter is activated on the following character boundary.

3.7 SDLC MODE

The one remaining synchronous mode available in the SCC is SDLC mode. SDLC mode is useful in bit-oriented protocols. That is, protocols which use the technique

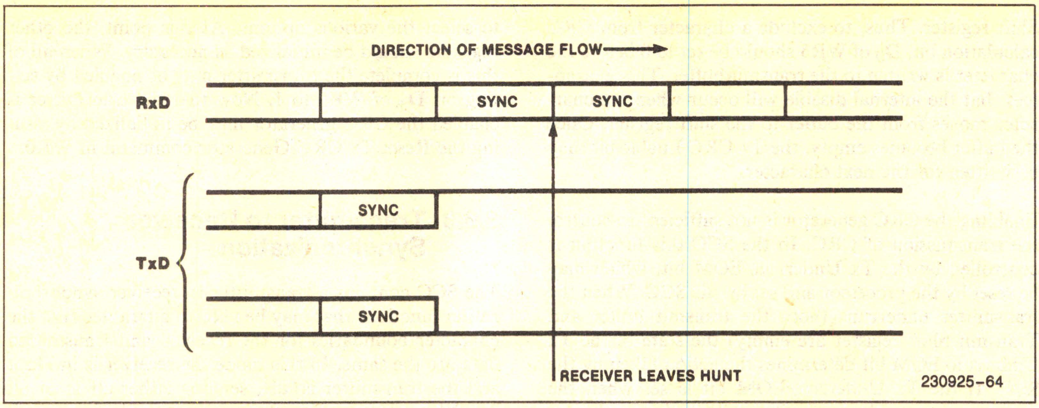


Figure 3-21. Transmitter to Receiver Synchronization

of 0 insertion to achieve data transparency. In SDLC mode, frames of information are opened and closed by a unique bit pattern called a flag (see Figure 3-22). The Flag character has a bit pattern of 01111110 and this sequence is unique because all data between the opening and closing flags is prohibited from having more than five consecutive 1s. The transmitter guarantees this by watching the transmit data stream and inserting a 0 after five consecutive ones, irrespective of character boundaries. In turn, the receiver searches the receive data stream for five consecutive 1s and deletes the next bit if it is a 0. CRC may be used in SDLC mode but only with the CRC-CCITT polynomial. This is because the transmitter in the SCC automatically inverts the CRC before transmission and the receiver, to compensate for this, checks the CRC result for the bit pattern 0001110100001111. This is consistent with bit-oriented protocols such as SDLC, HDLC and ADCCP. There are two unique bit patterns in SDLC mode besides the flag sequence. They are the Abort and EOP (End of Poll) sequence. An Abort is a sequence of from seven to thirteen consecutive 1s and is used to signal the premature termination of a frame. The EOP is the bit pattern 11111110, which is used in loop applications as a signal to a secondary station that it may begin transmission.

SDLC mode is selected by setting bit D_5 of WR4 to 1 and bits D_4, D_3, and D_2 of WR4 to 0. In addition, the flag sequence must be written to WR7. Additional control bits for SDLC mode are located in WR10.

3.7.1 SDLC Receive

The receiver in the SCC always searches the receive data stream, for flag characters in SDLC mode. Ordinarily, the receiver transfers all received data between flags to the receive data FIFO. However, if the receiver is in Hunt mode no flag is received. The receiver is in Hunt mode when first enabled or the receiver may be placed in Hunt mode by the processor. This is accomplished by issuing the enter Hunt mode command in WR3. This bit (D_4) is a command, so writing a 0 to it has no effect. The Hunt status of the receiver is reported by the Sync/Hunt bit in RR0. Sync/Hunt is one of the possible sources of external/status interrupts, with both transitions causing an interrupt. This is true even if the Sync/Hunt bit is set as a result of the processor issuing the Enter Hunt Mode command. The receiver will automatically Enter Hunt Mode if an abort is received. Because the receiver always searches the receive data stream for flags and automatically enters Hunt when an abort is received the receiver will always handle frames correctly and the Enter Hunt Mode command should never be needed. The SCC will drive the $\overline{\text{SYNC}}$ pin Low to signal that a flag has been recognized. The timing for the $\overline{\text{SYNC}}$ signal is shown in Figure 3-18.

The first byte in an SDLC frame is assumed by the SCC to be the address of the secondary station for which the frame is intended and the SCC provides several options for handling this address. If the Address Search Mode bit (D_2) in WR3 is set to 0 the address recognition logic is disabled and all received frames are transferred to the receive data FIFO. In this mode the software must perform any address recognition. If the Address Search Mode bit is set to 1, only those frames whose address matches the address programmed in WR6 or the global address (all 1s) will be transferred to the receive data FIFO. The address comparison will be across all eight

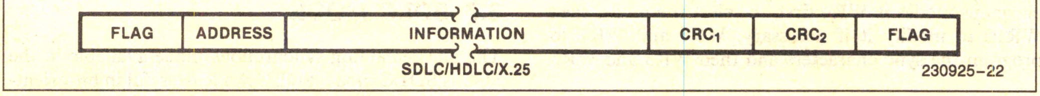


Figure 3-22. SDLC Message Format

bits of WR6 if the Sync Character Load Inhibit bit (D_1) in WR3 is set to 0. The comparison may be modified so that only the four most significant bits of WR6 must match the received address. This mode is selected by setting the Sync Character Load Inhibit bit to one. In this mode, however, the address field is still eight bits wide. The address field is transferred to the receive data FIFO in the same manner as data. That is, it is not treated differently than data.

The number of bits per character is controlled by bits D_7 and D_6 of WR3. Five, six, seven, or eight bits per character may be selected via these two bits. The data is right-justified in the receive buffer. The SCC merely takes a snapshot of the receive data stream at the appropriate times so the "unused" bits in the receive buffer are only the bits following the character in the data stream. An additional bit carrying parity information may be selected by setting bit D_6 of WR4 to 1. This also enables parity in the transmitter. The parity sense is selected by bit D_1 of WR4. Parity is not normally used in SDLC mode. The character length may be changed at any time before the new number of bits have been assembled by the receiver. Care should be exercised, however, as unexpected results may occur. A representative example, switching from five bits to eight bits and back to five bits is shown in Figure 3-19.

Most bit-oriented protocols allow an arbitrary number of bits between opening and closing Flags. The SCC allows for this by providing three bits of Residue Code in RR1 that indicate which bits in the last few bytes transferred from the receive data FIFO by the processor are actually valid data bits. The meaning of these three bits with each character length option is shown in Table 3-4. As indicated in the table, these bits allow the processor to determine those bits in the information (and not CRC) field. This allows transparent retransmission of the received frame. The Residue Code bits do not go through a FIFO so they change in RR1 when the last character of the frame is loaded into the receive data FIFO. If there are any characters already in the receive data FIFO the Residue Code will be updated before they are read by the processor. Thus these three bits of RR1 should be ignored by the processor unless the End of Frame bit in RR1 is set.

Only the CRC-CCITT polynomial may be used for CRC calculation in SDLC mode, although the generator and checker may be preset to all 1s or all 0s. The CRC-CCITT polynomial is selected by setting bit D_2 of WR5 to 0 and bit D_7 of WR10 controls the preset value. If this bit is set to 1, the generator and checker are preset to ones and, if this bit is reset, the generator and checker are preset to all 0s. The receiver expects the CRC to be inverted before transmission and so checks the CRC result against the value 0001110100001111. The SCC presets the CRC checker whenever the receiver is in Hunt mode or whenever a flag is received so a CRC reset command is not strictly necessary. However, the CRC checker may be preset by issuing the Reset CRC Checker command in WR0. This command is encoded in bits D_7 and D_6 of WR0. The CRC checker is automatically enabled for all data between the opening and closing flags by the SCC in SDLC mode, and the Rx CRC Enable bit (D_3) in WR3 is ignored. The result of the CRC calculation for the entire frame is valid in RR1 only when accompanied by the End of Frame bit being set in RR1. At all other times the CRC Error bit in RR1 should be ignored by the processor. Care must be exercised so that the processor does not attempt to use the CRC bytes that are transferred as data because not all of the bits are transferred properly. The last two bits of CRC are never transferred to the receive data FIFO and are not recoverable.

A frame is terminated by a closing flag and when the SCC recognizes this flag the contents of the Receive Shift register are transferred to the receive data FIFO, the Residue Code is latched, the CRC Error bit is latched in the status FIFO and the End Of Frame bit is set in the receive status FIFO. The End Of Frame bit, upon reaching the top of the FIFO, will cause a special receive condition. The processor may then read RR1 to

Table 3-4. Residue Codes

Residue Code			Bits in Previous Byte				Bits in Second Previous Byte				Bits in Third Previous Byte			
0	1	2	8B/C	7B/C	6B/C	5B/C	8B/C	7B/C	6B/C	5B/C	8B/C	7B/C	6B/C	5B/C
1	0	0	0	0	0	0	3	1	0	0	8	7	5	2
0	1	0	0	0	0	0	4	2	0	0	8	7	6	3
1	1	0	0	0	0	0	5	3	1	0	8	7	6	4
0	0	1	0	0	0	0	6	4	2	0	8	7	6	5
1	0	1	0	0	0	0	7	5	3	1	8	7	6	5
0	1	1	0	0	0	—	8	6	4	—	8	7	6	—
1	1	1	1	0	—	—	8	7	—	—	8	7	—	—
0	0	0	2	—	—	—	8	—	—	—	8	—	—	—

determine the result of the CRC calculation as well as the Residue Code. If either the Rx Interrupt On Special Condition Only or the Rx Interrupt on First Character or Special Condition modes are selected, the processor must issue an Error Reset command in WR0 to unlock the receive FIFO.

In addition to searching the data stream for flags, the receiver in the SCC also watches for seven consecutive ones, which is the abort condition. The presence of seven consecutive 1s is reported in the Break/Abort bit in RR0. This is one of the possible external/status interrupts so transitions of this status may be programmed to cause interrupts. Upon receipt of an abort the receiver is forced into Hunt mode, where it looks for flags. The hunt status is also a possible external/status condition whose transition may be programmed to cause an interrupt. The transitions of these two bits occur very close together but either one or two external/status interrupts may result. The abort condition is terminated when a 0 is received, either by itself or as the leading 0 of a flag. The receiver does not leave Hunt mode until a flag has been received so two discrete external status conditions will occur at the end of an abort. An abort received in the middle of a frame terminates the frame reception, but not in an orderly manner, because the character being assembled is lost.

Up to two modem control signals associated with the receiver are available in SDLC mode. The $\overline{DTR/REQ}$ pin carries the inverted state of the DTR bit (D_7) in WR5 unless this pin has been programmed to carry a DMA Request signal. The $\overline{CD}$ pin is ordinarily a simple input to the CD bit in RR0. However, if the Auto Enables mode is selected by setting bit D_5 of WR3 to 1, this pin becomes an enable for the receiver. That is, if Auto Enables is on and the $\overline{CD}$ pin is High the receiver is disabled, while the receiver is enabled while the $\overline{CD}$ pin is Low.

The initialization sequence for the receiver in SDLC mode is WR4 first, to select the mode, then WR10 to modify it if necessary, WR6 to program the address, WR7 to program the flag and then WR3 and WR5 to select the various options. At this point the other registers should be initialized as necessary. When all of this is complete the receiver may be enabled by setting bit D_6 of WR3 to 1.

3.7.2 SDLC Transmit

Once SDLC mode has been selected the flag must be written in WR7, to be used to open and close the transmitted frames. The SCC does not automatically send the address byte; it merely encapsulates the data supplied by the processor with flags and CRC. Ordinarily, a frame will be terminated by the SCC with CRC and a flag but the SCC may be programmed to send an abort and a flag in place of the CRC. This option allows the SCC to abort a frame transmission in progress if the transmitter is accidentally allowed to underrun. This is controlled by the Abort/$\overline{Flag}$ On Underrun bit (D_2) in WR10. When this bit is set to 1 the transmitter will send an abort and a flag in place of the CRC when an underrun occurs. The frame will be terminated normally, with CRC and a flag, if this bit is set to 0. The SCC is also able to send an abort by command of the processor. The Send Abort command, issued in WR0, will send eight consecutive 1s and then the transmitter will idle. Since up to five consecutive 1s may have been sent prior to the command being issued, a Send Abort will cause a sequence of from eight to thirteen 1s to be transmitted. The Send Abort command also empties the transmit buffer register. The idle condition for the transmitter is continuous flags, but this is under program control. By setting the Mark/$\overline{Flag}$ Idle bit (D_3) in WR10 to 1, the transmitter will send continuous 1s in place of the idle flags. Note that the closing flag will be transmitted correctly even if this mode is selected. The Mark/$\overline{Flag}$ Idle must be set to 0, allowing a flag to be transmitted, before data is written to the transmit buffer. Care must be exercised in doing this because the continuous 1s are transmitted eight at a time, and all eight must leave the Transmit Shift register so that a flag may be loaded into it before the first data is written to the transmit buffer. When using the transmitter in SDLC mode recall that all data passes through the zero inserter, which adds an extra five bit times of delay between the Transmit Shift register and the Transmit Data pin.

The number of bits per transmitted character is controlled by bits D_6 and D_5 of WR5 and the way the data is formatted within the transmit buffer. The bits in WR5 allow the option of five, six, seven or eight bits per character. When five bits per character is selected the data may be formatted before being written to the transmit buffer to allow transmission of from one to five bits per character. This formatting is shown in Table 3-3. In all cases the data must be right-justified, with the unused bits being ignored except in the case of five bits per character.

An additional bit, carrying parity information, may be automatically appended to every transmitted character by setting bit D_6 of WR4 to 1. This bit is sent in addition to the number of bits specified in WR4 or by the data format. The parity sense is selected by bit D_1 of WR4. Parity is not normally used in SDLC mode. The character length may be changed on the fly, but the desired length must be selected before the character is loaded into the transmit shift register from the transmit buffer. The easiest way to ensure this is to write to WR5 to change the character length before writing the data to the transmit buffer.

Only the CRC-CCITT polynomial may be used in SDLC mode. This is selected by setting bit D_2 in WR5 to 0. This bit controls the selection for both the transmitter and receiver. The initial state of the generator and checker is controlled by bit D_7 of WR10. When

this bit is set to 1, both the generator and checker will have an initial value of all 1s and, if this bit is set to 0, the initial values will be all 0s. The SCC does not automatically preset the CRC generator so this must be done in software. This is accomplished by issuing the Reset Tx CRC Generator command, which is encoded in bits D_7 and D_6 of WR0. For proper results, this command must be issued while the transmitter is enabled and idling. If CRC is to be used the transmit CRC generator must be enabled by setting bit D_0 of WR5 to 1. CRC is normally calculated on all characters between opening and closing flags so this bit is usually set to 1 at initialization and never changed.

Enabling the CRC generator is not sufficient to control the transmission of CRC. In the SCC this function is controlled by the Tx Underrun/EOM bit, which may be reset by the processor and set by the SCC. When the transmitter underruns (both the transmit buffer and Transmit Shift register are empty) the state of the Tx Underrun/EOM bit determines the action taken by the SCC. If the Tx Underrun/EOM bit is set when the underrun occurs, the transmitter will send flags; if this bit is reset when the underrun occurs, the transmitter will send either the accumulated CRC followed by flags, or an abort followed by flags, depending on the state of the Abort/$\overline{\text{FLAG}}$ on Underrun bit in WR10. When the CRC or abort is loaded into the Transmit Shift register for transmission, the SCC will set the Tx Underrun/EOM bit to indicate this. This transition may be programmed to cause an external/status interrupt, or the Tx Underrun/EOM bit is available in RR0. The Reset Tx Underrun/EOM Latch command is encoded in bits D_7 and D_6 of WR0. For correct transmission of the CRC at the end of a frame, this command must be issued after the first character is written to the SCC but before the transmitter underruns after the last character written to the SCC. The command is usually issued immediately after the first character is written to the SCC so that the abort or CRC is sent if an underrun occurs inadvertently. The Abort/Flag on Underrun bit (D_2) in WR10 is usually set to 1 at the same time as the Tx Underrun/EOM bit is reset so that an abort will be sent if the transmitter underruns. The bit is then set to 0 near the end of the frame to allow the correct transmission of CRC.

In this paragraph the term "completely sent" means shifted out of the Transmit Shift register, not shifted out of the zero inserter, which is an additional five bit times of delay. In SDLC mode, if the transmitter is disabled during transmission of a character, that character will be "completely sent." This applies to both data and flags. However, if the transmitter is disabled during the transmission of CRC, the 16-bit transmission will be completed but the remaining bits will be from the Flag register rather than the remainder of the CRC.

There are two modem control signals associated with the transmitter provided by the SCC. The $\overline{\text{RTS}}$ pin is a simple output that carries the inverted state of the RTS bit (D_1) in WR5. The $\overline{\text{CTS}}$ pin is ordinarily a simple input to the CTS bit in RR0. However, if Auto Enables mode is selected this pin becomes an enable for the transmitter. That is, if Auto Enables is on and the $\overline{\text{CTS}}$ pin is High the transmitter is disabled, while the transmitter is enabled if the $\overline{\text{CTS}}$ pin is Low.

The initialization sequence for the transmitter in SDLC mode is WR4 first, to select the mode, then WR10 to modify it if necessary, WR7 to program the flag and then WR3 and WR5 to select the various options. At this point the other registers should be initialized as necessary. When all of this is complete the transmitter may be enabled by setting bit D_3 of WR5 to 1. Now that the transmitter is enabled the CRC generator may be initialized by issuing the Reset Tx CRC Generator command in WR0.

3.7.3 SDLC Loop Mode

SDLC Loop mode is quite similar to SDLC mode except that two additional control bits are used. They are the Loop Mode bit (D_1) and the Go Active On Poll bit (D_4) in WR10. In addition to the two extra control bits in SDLC Loop mode, there are also two status bits in RR10. They are the On Loop bit (D_1) and the Loop Sending bit (D4). Before Loop mode is selected both the receiver and transmitter must be completely initialized for SDLC operation. Once this is done Loop mode is selected by setting bit D_1 of WR10 to 1. At this point the SCC connects TxD to RxD with only gate delays in the path. At the same time a flag is loaded into the Transmit Shift register and is shifted to the end of the zero inserter, ready for transmission. The SCC will remain in this state until the Go Active On Poll bit (D_4) in WR10 is set to 1. When this bit is set to 1 the receiver begins looking for a sequence of seven consecutive ones, indicating either an EOP or an idle line. When the receiver detects this condition the Break/Abort bit in RR0 is set to 1 and a one-bit time delay is inserted in the path from RxD to TxD. The On Loop bit in RR10 is also set to 1 at this time, and the receiver enters the Hunt mode. The SCC cannot transmit on the loop until a flag is received, causing the receiver to leave Hunt mode, and another EOP (bit pattern 11111110) is received. The SCC is now on the loop and capable of transmitting on the loop. As soon as this status is recognized by the processor the Go Active On Poll bit in WR10 should be set to 0 to prevent the SCC from transmitting on the loop without the consent of the processor.

To transmit a message on the loop the Go Active On Poll bit in WR10 must be set to 1. Once this is done the SCC will change the next received EOP into a Flag and begin transmitting on the loop. When the EOP is received the Break/Abort and Hunt bits in RR0 will be set to 1 and the Loop Sending bit in RR10 will also be set to 1. Data to be transmitted may be written after the

Go Active On Poll bit has been set or after the receiver enters Hunt mode. If the data is written immediately after the Go Active On Poll bit is set, the SCC will only insert one flag after the EOP is changed into a flag. If the data is not written until after the receiver enters the Hunt mode, flags will be transmitted until the data is written. If only one frame is to be transmitted on the loop in response to an EOP the processor must set the Go Active On Poll bit to 0 before the last data is written to the transmitter. In this case the transmitter will close the frame with a single flag and then revert to the one-bit delay. The Loop Sending bit in RR10 is set to 0 when the closing Flag has been sent. If more than one frame is to be transmitted the Go Active On Poll bit should not be set to 0 until the last frame is being sent. If this bit is not set to 0 before the end of a frame, the transmitter will send Flags until either more data is written to the transmitter or until the Go Active On Poll bit is set to 0. Note that the state of the Abort/$\overline{\text{Flag}}$ on Underrun and Mark/$\overline{\text{Flag}}$ Idle bits in WR10 are ignored by the SCC in SDLC Loop mode.

To go off of the loop in an orderly manner requires actions similar to those taken to go on the loop. First, the Go Active On Poll bit must be set to 0 and any transmission in progress completed. This is if the SCC is currently sending on the loop. Once the SCC is not currently sending on the loop, an exit from the loop is accomplished by setting the Loop Mode bit in WR10 to 0, and at the same time writing the Abort/$\overline{\text{Flag}}$ on Underrun and Mark/$\overline{\text{Flag}}$ Idle bits with the desired values. The SCC will revert to normal SDLC operation as soon as an EOP is received, or immediately, if the receiver is already in Hunt mode because of the receipt of an EOP.

The initialization sequence for the SCC in SDLC Loop mode is similar to the sequence used in SDLC mode, except that it is somewhat longer. The processor should program WR4 first, to select SDLC mode, and then WR10 to select the CRC preset value and program the Mark/$\overline{\text{Flag}}$ Idle bit. The Loop Mode and Go Active On Poll bits in WR10 should not be set to 1 yet. The flag is written in WR7 and the various options are selected in WR3 and WR5. At this point the other registers should be initialized as necessary. At this point the Loop Mode bit (D_1) in WR10 should be set to 1. When all of this is complete the transmitter may be enabled by setting bit D_3 of WR5 to 1. Now that the transmitter is enabled the CRC generator may be initialized by issuing the Reset Tx CRC Generator command in WR0. The receiver is enabled by setting bit D_1 of WR3 to 1. At this point the SCC is waiting for permission to go on the loop. The processor grants permission for this by setting the Go Active On Poll bit (D_4) in WR10 to 1. The SCC will go on the loop when seven consecutive 1s are received and will signal this by setting the On Loop bit in RR10. Note that the seven consecutive 1s will set the Break/Abort and Hunt bits in RR0 also. Once the SCC is on the loop the go Active On Poll bit should be set to 0 until a message is to be transmitted on the loop. To transmit a message on the loop the Go Active On Poll bit should be set to 1. At this point the processor may either write the first character to the transmit buffer and wait for a transmit buffer empty condition, or wait for the Break/Abort and Hunt bits to be set in RR0 and the Loop Sending bit to be set in RR10 before writing the first data to the transmitter. The Go Active On Poll bit should be set to 0 after the transmission of the frame has begun. To go off of the loop the processor should set the Go Active On Poll bit in WR10 to 0 and then wait for the Loop Sending bit in RR10 to be set to 0. At this point the Loop Mode bit (D_1) in WR10 is set to 0 to request an orderly exit from the loop. The SCC will exit SDLC Loop mode when seven consecutive 1s have been received, and at the same time the Break/Abort and Hunt bits in RR0 will be set to 1 and the On Loop bit in RR10 will be set to 0.

3.8 BAUD RATE GENERATOR

A block diagram of the baud rate generator is shown in Figure 3-23. It consists of a 16-bit downcounter, two 8-bit time constant registers and an output divide-by-two. The baud rate generator input comes from the output of a two-input multiplexer and zero count condition is output to the external/status interrupt section. The baud rate generator may be enabled and disabled by command and is disabled by a hardware reset.

The time constant for the baud rate generator is programmed in WR12 and WR13, with the least-significant byte being held in WR12. The formulas relating the baud rate to the time constant and vice versa are shown in Table 3-5 along with an example. In these formulas the baud rate generator clock frequency is in Hertz, the desired baud rate is in bits/second and the time constant is dimensionless. The example in Table 3-5 assumes a 3.9936 MHz clock frequency and shows the time constants for a number of popular baud rates.

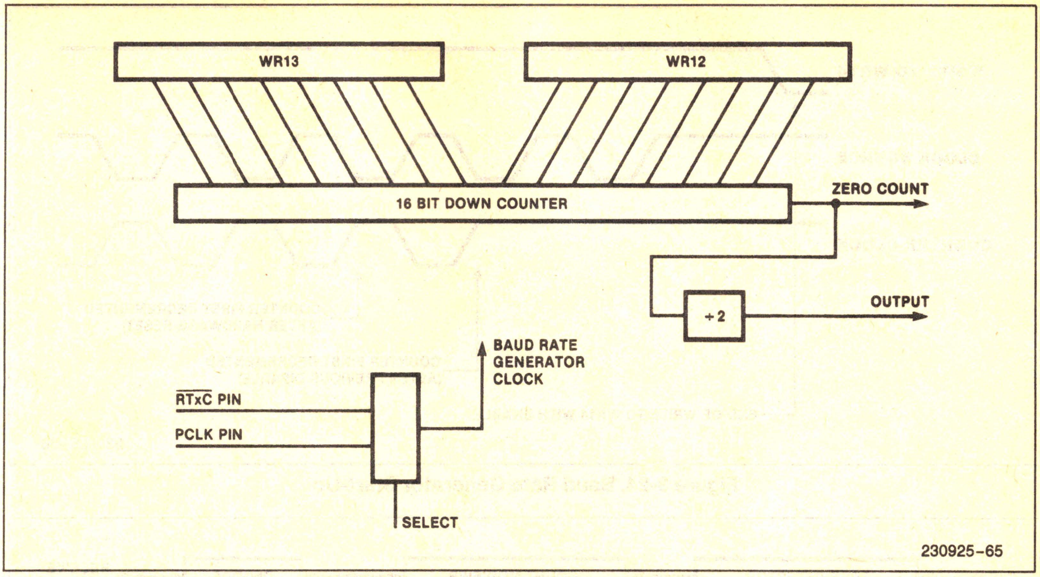


Figure 3-23. Baud Rate Generator

Table 3-5. Time Constant Formulas

$$\text{Time Constant} = \left[\frac{\text{Clock Frequency}}{2 * (\text{Baud Rate})}\right] - 2$$

$$\text{Baud Rate} = \frac{\text{Clock Frequency}}{2 * (\text{Time Constant} + 2)}$$

Rate	3.9936 MHz Source		Error
	Time	Constant	
19200	102	(%0066)	0
9600	206	(%00CE)	0
7200	275	(%0113)	0.12%
4800	414	(%019E)	0
3600	553	(%0229)	0.06%
2400	830	(%033E)	0
2000	996	(%03E4)	0.04%
1800	1107	(%0453)	0.03%
1200	1662	(%067E)	0
600	3326	(%0CFE)	0
300	6654	(%19FE)	0
150	13310	(%33FE)	0
134.5	14844	(%39FC)	0.0007%
110	18151	(%46E7)	0.0015%
75	26622	(%67FE)	0
50	39934	(%9BFE)	0

The clock source for the baud rate generator is selected by bit D_1 of WR14. When this bit is set to 0 the baud rate generator uses the signal on the $\overline{\text{RTxC}}$ pin as its clock, independent of whether the $\overline{\text{RTxC}}$ pin is a simple input or part of the crystal oscillator circuit. When this bit is set to 1 the baud rate generator is clocked by CLK. To avoid metastable problems in the counter, this bit should be changed only while the baud rate generator is disabled, since arbitrarily narrow pulses can be generated at the output of the multiplexer when it changes status.

The baud rate generator is enabled while bit D_0 of WR14 is set to 1 and is disabled while this bit is set to 0. To prevent metastable problems when the baud rate generator is first enabled, the enable bit is synchronized to the baud rate generator clock. This introduces an additional delay when the baud rate generator is first enabled and this is shown in Figure 3-24. The baud rate generator is disabled immediately when bit D_0 of WR14 is set to 0, because the delay is only necessary on startup. The baud rate generator may be enabled and disabled on the fly, but this delay on startup must be taken into consideration.

Upon reaching a count of 0 the time constant held in WR12 and WR13 is reloaded into the downcounter so that the process of counting down may start over. In addition to reloading the time constant, the output of the baud rate generator toggles, and for the clock cycle with a zero count, the zero count signal goes active to the external/status section. This zero count condition from the baud rate generator does not persist, so if it is to be used by the processor, it should be latched in the external/status latch. While the baud rate generator is disabled the state of the zero count signal is held. This signal is forced active by a hardware reset.

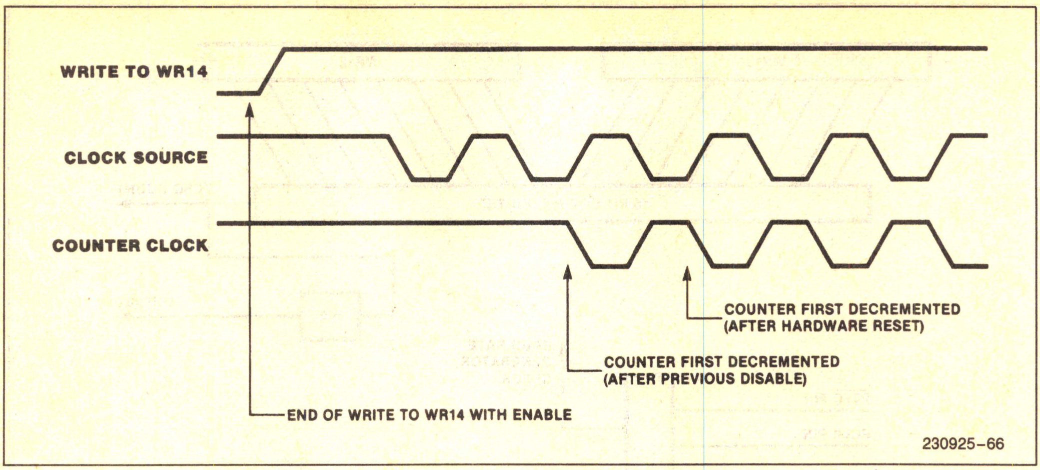


Figure 3-24. Baud Rate Generator Start-Up

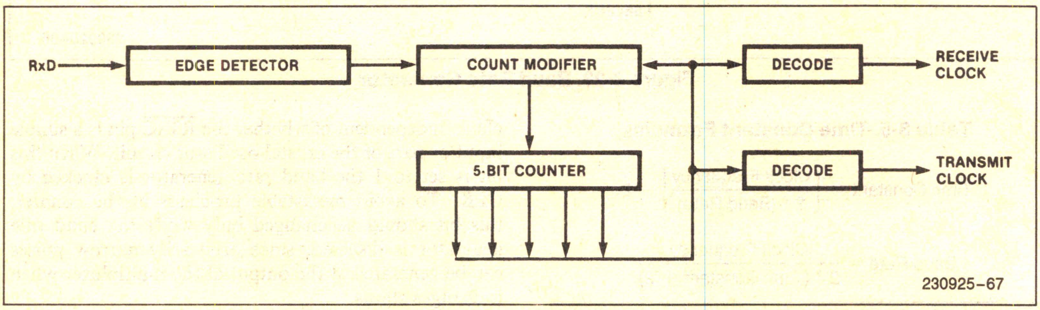


Figure 3-25. Digital Phase-Locked Loop

Initializing the baud rate generator is done in four steps. First, the time constant is determined and loaded into WR12 and WR13. Next, the processor should select the clock source for the baud rate generator by writing to bit D_1 of WR14. Finally, the baud rate generator is enabled by setting bit D_0 of WR14 to 1. Note that the first write to WR14 is not necessary after a hardware reset if the clock source is to be the $\overline{RTxC}$ pin. This is because a hardware reset automatically selects the $\overline{RTxC}$ pin as the baud rate generator clock source.

3.9 DIGITAL PHASE LOCKED LOOP

A block diagram of the digital phase locked loop is shown in Figure 3-25. It consists of a 5-bit counter, an edge detector, and a pair of output decoders. The clock for the DPLL comes from the output of a two input multiplexer and the two outputs go to the transmit and receive clock multiplexers. The DPLL is controlled by the seven commands that are encoded in bits D_7, D_6, and D_5 of WR14.

The clock for the DPLL is selected by two of the commands in WR14. One command selects the output of the baud rate generator as the clock source and the other command selects the $\overline{RTxC}$ pin as the clock source, independent of whether the $\overline{RTxC}$ pin is a simple input or part of the crystal oscillator circuit. To avoid metastable problems in the counter, the clock source selection should be made only while the DPLL is disabled, since arbitrarily narrow pulses can be generated at the output of the multiplexer when it changes status.

The DPLL is enabled by issuing the Enter Search Mode command in WR14. This command is also used to reset the DPLL to a known state if it is suspected that synchronization has been lost. When used to enable the DPLL the Enter Search Mode command unlocks the counter, which is held while the DPLL is disabled, and enables the edge detector. The DPLL is now in Search mode. If the DPLL is already enabled when this command is issued the DPLL also enters Search mode. While in Search mode, the counter is

held at a specific count and no outputs are provided. The DPLL remains in this status until an edge is detected in the receive data stream. This first edge is assumed to occur on a bit cell boundary and the DPLL will begin providing an output to the receiver that will properly sample the data. From this point on the DPLL will adjust its output to remain in phase with the receive data. If the first edge that the DPLL sees does not occur on a bit cell boundary, the DPLL will still lock on to the receive data although it will take longer to do so.

The DPLL may be programmed to operate in either of two modes, as selected by command in WR14. In the NRZI mode the DPLL clock must be 32 times the data rate. In this mode the transmit and receive clock outputs of the DPLL are identical, with the clocks phased so that the receiver samples the data in the middle of the bit cell. In NRZI mode the DPLL does not require a transition in every bit cell, so this mode is useful for recovering the clocking information from NRZ and NRZI data streams. In the FM mode the DPLL clock must be 16 times the data rate. In this mode the transmit clock output of the DPLL lags the receive clock output by 90°, to make the transmit and receive bit cell boundaries the same because the receiver must sample FM data at one-quarter and three-quarters bit time. In FM mode the DPLL requires a transition in every bit cell, and if this transition is not present in two consecutively sampled bit cells the DPLL will automatically enter the search mode. As in the case of the clock source selection, the mode of operation should only be changed while the DPLL is disabled to prevent unpredictable results.

3.9.1 NRZI Mode Operation

To operate in NRZI mode the DPLL must be supplied with a clock that is 32 times the data rate. The DPLL uses this clock, along with the receive data, to construct receive and transmit clock outputs that are phased to properly receive and transmit data. To do this, the DPLL divides each bit cell into four regions, and makes an adjustment to the count cycle of the 5-bit counter dependent upon which region a transition on the receive data input occurred. This is shown in Figure 3-26. Ordinarily, a bit cell boundary will occur between count 15 and count 16, and the DPLL output will cause the data to be sampled in the middle of the bit cell. The DPLL actually allows the transition marking a bit cell boundary to occur anywhere during the second half of count 15 or the first half of count 16 without making a correction to its count cycle. However, if the transition marking a bit cell boundary occurs between the middle of count 16 and count 31 the output

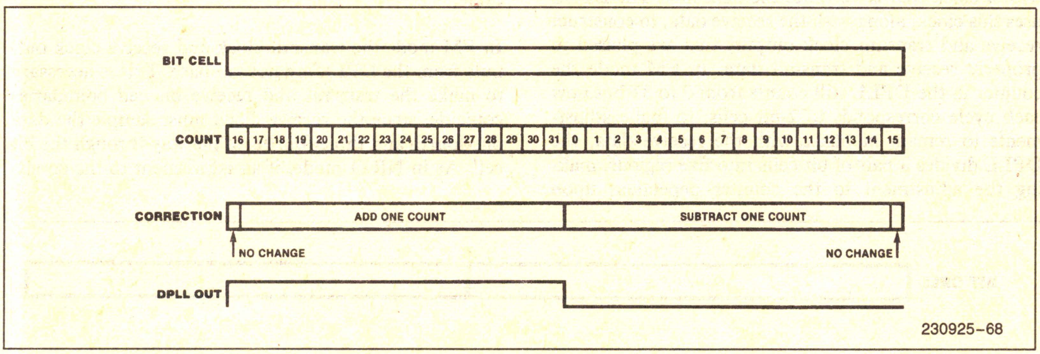


Figure 3-26. DPLL in NRZI Mode

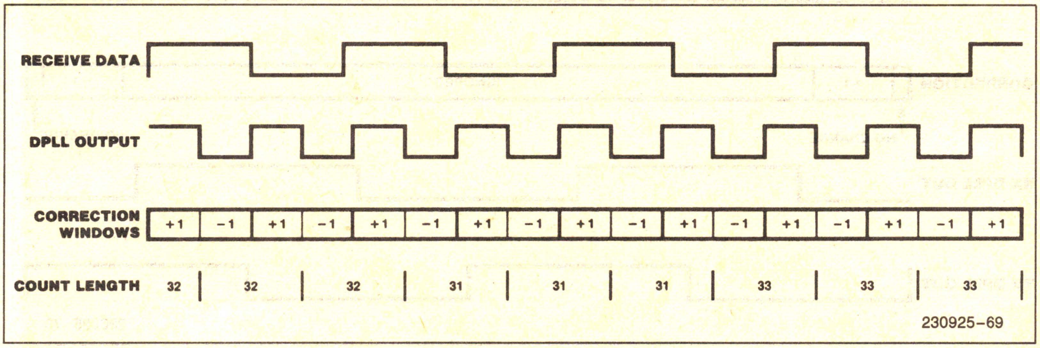


Figure 3-27. DPLL Operating Example

of the DPLL is causing the data to be sampled too early in the bit cell. In response to this the DPLL extends its count by one during the next 0 to 31 counting cycle, which effectively moves the edge of the clock that samples the receive data closer to the center of the bit cell. In a similar manner, if the transition occurs between count 0 and the middle of count 15, the output of the DPLL is causing the data to be sampled too late in the bit cell. To correct this the DPLL shortens its count by one during the next 0 to 31 counting cycle, which effectively moves the edge of the clock that samples the receive data closer to the center of the bit cell. In NRZI mode, if the DPLL does not see any transition during a counting cycle, no adjustment is made in the following counting cycle. If an adjustment to the counting cycle is necessary the DPLL modifies count five, either deleting it or doubling it. Thus only the Low time of the DPLL output will be lengthened or shortened. While the DPLL is in search mode the counter remains at count 16, where the DPLL outputs are both High. The missing clock latches in the DPLL, which may be accessed in RR10, are not used in NRZI mode. An example of the DPLL in operation is shown in Figure 3-27.

3.9.2 FM Mode Operation

To operate in FM mode the DPLL must be supplied with a clock that is 16 times the data rate. The DPLL uses this clock, along with the receive data, to construct receive and transmit clock outputs that are phased to properly receive and transmit data. In FM mode the counter in the DPLL still counts from 0 to 31 but now each cycle corresponds to 2-bit cells; to make adjustments to remain in phase with the receive data, the DPLL divides a pair of bit cells into five regions, making the adjustment to the counter dependent upon which region the transition on the receive data input occurred. This is shown in Figure 3-28. Ordinarily a bit cell boundary will occur between count fifteen and count sixteen, and the DPLL receive output will cause the data to be sampled at one-fourth and three-fourths of the way through the bit cell. The DPLL actually allows the transition marking a bit-cell boundary to occur anywhere during the second half of count 15 or the first half of count 16 without making a correction to its count cycle. However, if the transition marking a bit cell boundary occurs between the middle of count 16 and the middle of count 19 the output of the DPLL is causing the data to be sampled too early in the bit cell. In response to this the DPLL extends its count by 1 during the next 0 to 31 counting cycle, which effectively moves the receive clock edges closer to where they should be. In a similar manner, if the transition occurs betwen the middle of count 12 and the middle of count 15, the output of the DPLL is causing the data to be sampled too late in the bit cell. To correct this the DPLL shortens its count by 1 during the next 0 to 31 counting cycle, which effectively moves the receive clock edges closer to where they should be. In FM mode any transitions occurring between the middle of count 19 in one cycle, and the middle of count 12 during the next cycle, are ignored by the DPLL. This is necessary to guarantee that any data transitions in the bit cells will not cause an adjustment to the counting cycle.

In FM mode the transmit clock and receive clock outputs from the DPLL are not in phase. This is necessary to make the transmit and receive bit cell boundaries coincide, since the receive clock must sample the data one-fourth and three-fourths of the way through the bit cell. As in NRZI mode, if an adjustment to the count-

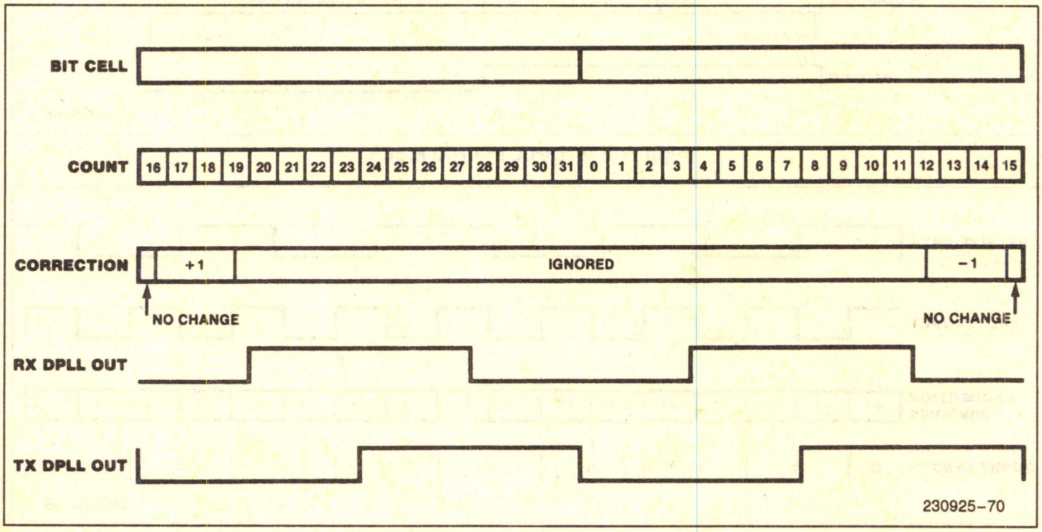


Figure 3-28. DPLL in FM Mode

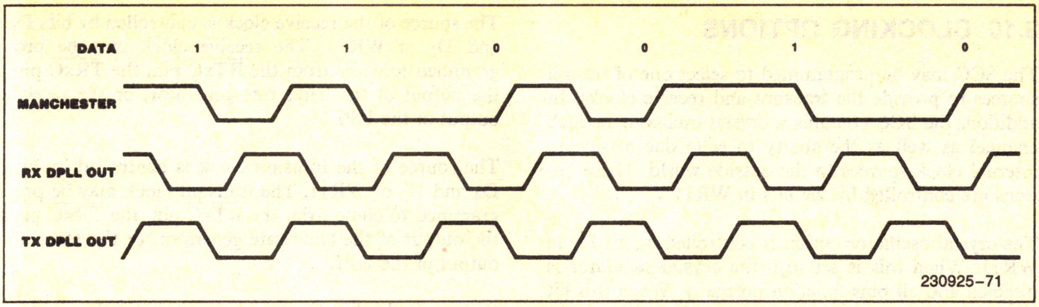


Figure 3-29. Manchester Clock Recovery

ing cycle is necessary the DPLL modifies count five, either deleting it or doubling it. If no adjustment is necessary the count sequence proceeds normally. While the DPLL is in search mode the counter remains at count 16, where the receive output is Low and the transmit output is Low. This fact can be used to provide a transmit clock under software control since the DPLL is in search mode while it is disabled. While the DPLL is disabled the transmit clock output of the DPLL may be toggled by alternately selecting FM and NRZI mode in the DPLL. The same is true of the receive clock.

In addition to FM encoded data, the DPLL may also be used to recover the clock from Manchester encoded data, which contains a transition at the center of every bit cell; it is the direction of this transition that distinguishes a 1 from a 0. Another way of looking at Manchester encoding is to realize that, during the first half of the bit cell, the data is sent; during the second half of the bit cell, the complement of the data is sent. This is shown in Figure 3-29, along with the DPLL output if it thinks that the mid-bit transitions are really bit cell boundaries. As is obvious from the figure, if the receiver samples the data on the falling edge of the DPLL receive clock output, the Manchester data will have been decoded. This occurs if the receiver is programmed to accept NRZ data.

From the above discussion together with an examination of FM0 and FM1 data encoding, it should be obvious that only clock transitions should exist on the receive data pin when the DPLL is programmed to enter search mode. If this is not the case the DPLL may attempt to lock on to the data transitions. With FM0 encoding this requires continuous ones received when leaving search. In FM1 encoding it is continuous 0s; with Manchester encoded data this means alternating 1s and 0s. With all three of these data encoding methods there will always be at least one transition in every bit cell, and in FM mode the DPLL is designed to expect this transition. In particular, if no transition occurs between the middle of count 12 and the middle of count 19, the DPLL is probably not locked onto the data properly. When the DPLL misses an edge the One Clock Missing bit is RR10 is set to 1 and latched. It will hold this value until a Reset Missing Clock command is issued in WR14 or until the DPLL is disabled or programmed to enter the Search mode. Upon missing this one edge the DPLL takes no other action and does not modify its count during the next counting cycle. However, if the DPLL does not see an edge between the middle of count 12 and the middle of count 19 in two successive 0 to 31 count cycles, a line error condition is assumed. If this occurs the two Clocks Missing bit in RR10 is set to 1 and latched. At the same time the DPLL enters the Search mode. The DPLL makes the decision to enter Search mode during count two, where both the receive clock and transmit clock outputs are Low. This prevents any glitches on the clock outputs when search mode is entered. While in search mode no clock outputs are provided by the DPLL. The Two Clocks Missing bit in RR10 is latched until a Reset Missing Clock command is issued in WR14 or until the DPLL is disabled or programmed to enter the Search mode.

3.9.3 DPLL Initialization

Initialization of the DPLL may be done at any time during the initialization sequence, but should probably be done after the clock modes have been selected in WR11 and before the receiver and transmitter are enabled. When initializing the DPLL the clock source should be selected first, followed by the selection of the operating mode. At this point the DPLL may be enabled by issuing the Enter Search Mode command in WR14. Note that a channel or hardware reset disables the DPLL, selects the $\overline{\text{RTxC}}$ pins as the clock source for the DPLL, and places it in the NRZI mode.

3.10 CLOCKING OPTIONS

The SCC may be programmed to select one of several sources to provide the transmit and receive clocks. In addition, the SCC contains a crystal oscillator in each channel as well as the ability to echo one of several internal clock sources to the outside world. These options are controlled by the bits in WR11.

The crystal oscillator option is controlled by bit D_7 in WR11. When this is set to 0 the crystal oscillator is disabled and all pins function normally. When this bit is set to 1 the crystal oscillator is enabled and a high-gain amplifier is connected between the $\overline{RTxC}$ pin and the $\overline{SYNC}$ pin. While the crystal oscillator is enabled, anything that has $\overline{RTxC}$ selected as its clock source will automatically be connected to the output of the crystal oscillator. While the crystal oscillator is enabled the $\overline{SYNC}$ pin is obviously unavailable for other use. Thus in synchronous modes no sync pulse is output, and the External Sync mode cannot be selected. In asynchronous modes the state of the Sync/Hunt bit in RR0 is no longer controlled by the $\overline{SYNC}$ pin. Instead, the Sync/Hunt bit is forced to 0. The crystal oscillator requires some finite time to stabilize. The oscillator must be allowed to stabilize before the output of the oscillator is used as a clock source.

The source of the receive clock is controlled by bits D_6 and D_5 of WR11. The receive clock may be programmed to come from the $\overline{RTxC}$ pin, the $\overline{TRxC}$ pin, the output of the baud rate generator, or the receive output of the DPLL.

The source of the transmit clock is controlled by bits D_4 and D_3 of WR11. The transmit clock may be programmed to come from the $\overline{RTxC}$ pin, the $\overline{TRxC}$ pin, the output of the baud rate generator, or the transmit output of the DPLL.

Ordinarily the $\overline{TRxC}$ pin is an input, but if this pin has not been selected as the source for the transmitter or the receiver, and bit D_2 of WR11 is set to 1, the $\overline{TRxC}$ pin becomes an output. The selection of the signal provided on the $\overline{TRxC}$ output pin is controlled by bits D_1 and D_0 of WR11. The $\overline{TRxC}$ pin may be programmed to provide the output of the crystal oscillator, the output of the baud rate generator, the receive output of the DPLL or the actual transmit clock. If the output of the crystal oscillator is selected but the crystal oscillator has not been enabled the $\overline{TRxC}$ pin will be driven High. The option of placing the transmit clock signal on the $\overline{TRxC}$ pin when it is an output allows access to the transmit output of the DPLL.

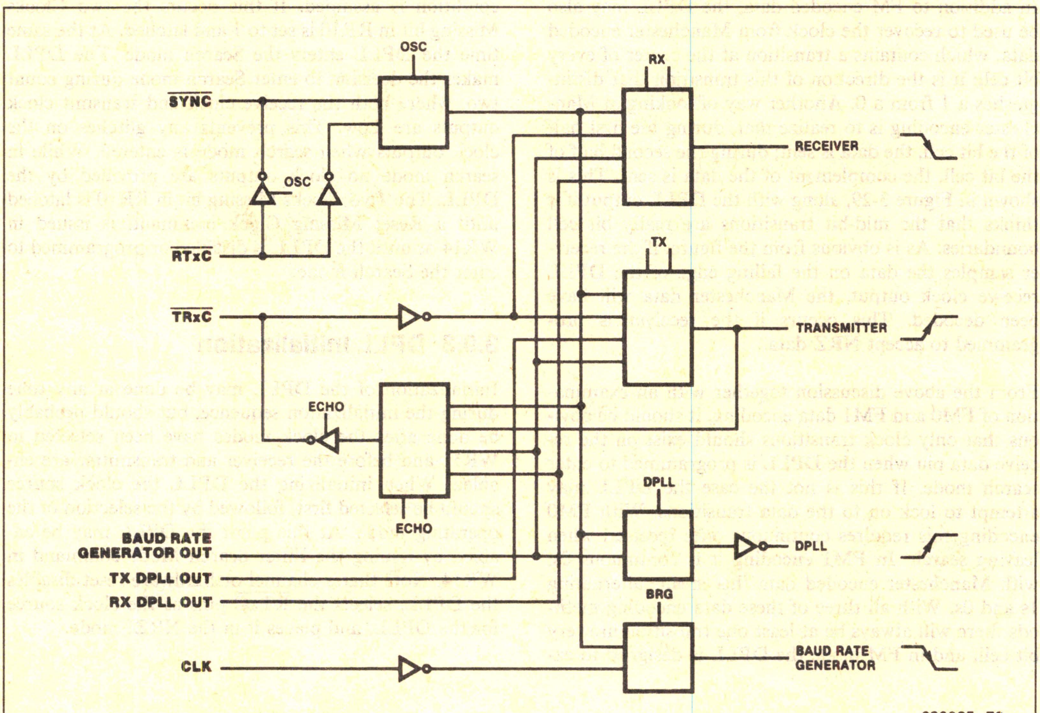


Figure 3-30. Clock Multiplexer

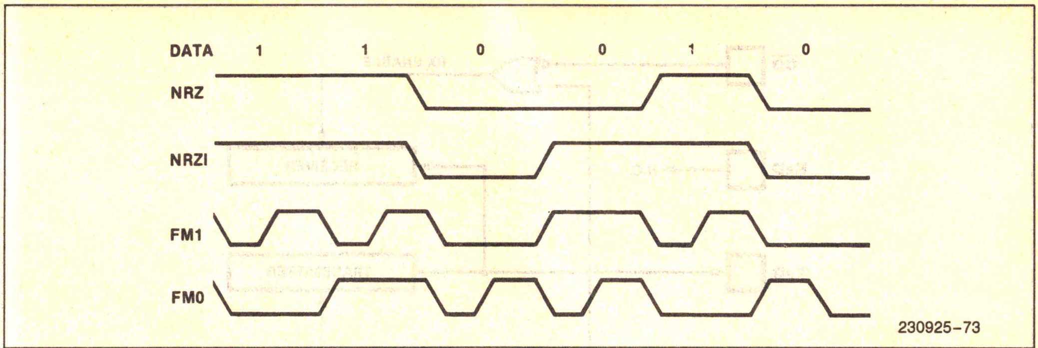


Figure 3-31. Data Encoding

A simplified schematic diagram of the circuitry used in the clock multiplexing is shown in Figure 3-30. This figure shows the inputs to the multiplexer section as well as the various signal inversions that occur in the paths to the outputs. Also shown are the edges used by the receiver, transmitter, baud rate generator and DPLL to sample or send data or otherwise change state. For example, the receiver samples data on the falling edge but since there is an inversion in the clock path between the $\overline{\text{RTxC}}$ pin and the receiver, a rising edge of the $\overline{\text{RTxC}}$ pin samples the data for the receiver.

Selection of the clocking options may be done anywhere in the initialization sequence, but it is better to have the final values selected before the receiver, transmitter, baud rate generator, or DPLL are enabled to prevent problems from arbitrarily narrow clock signals out of the multiplexers. The same is true of the crystal oscillator, in that the output should be allowed to stabilize before it is used as a clock source.

3.11 DATA ENCODING

The SCC provides four different data encoding methods, selected by bits D_6 and D_5 in WR10. An example of these four encoding methods is shown in Figure 3-31. Any encoding method may be used in any 1X mode in the SCC, asynchronous or synchronous. The data encoding selected is active even though the transmitter or receiver may be idling or disabled.

In NRZ encoding a 1 is represented by a High level and a 0 is represented by a Low level. In this encoding method only a minimal amount of clocking information is available in the data stream in the form of transitions on bit-cell boundaries. In an arbitrary data pattern this may not be sufficient to generate a clock for the data from the data itself.

In NRZI encoding a 1 is represented by no change in the level and a 0 is respresented by a change in the level. As in NRZ only a minimal amount of clocking information is available in the data stream, in the form of transitions on bit cell boundaries. In an arbitrary data pattern this may not be sufficient to generate a clock for the data from the data itself. In the case of SDLC, where the number of consecutive 1s in the data stream is limited, sufficient transitions to generate a clock are guaranteed.

In FM1 encoding, also known as biphase mark, a transition is present on every bit cell boundary and an additional transition may be present in the middle of the bit cell. In FM1 a 0 is sent as no transition in the center of the bit cell and a 1 is sent as a transition in the center of the bit cell. Because of all the transitions, FM1 encoded data contains sufficient information to recover a clock from the data.

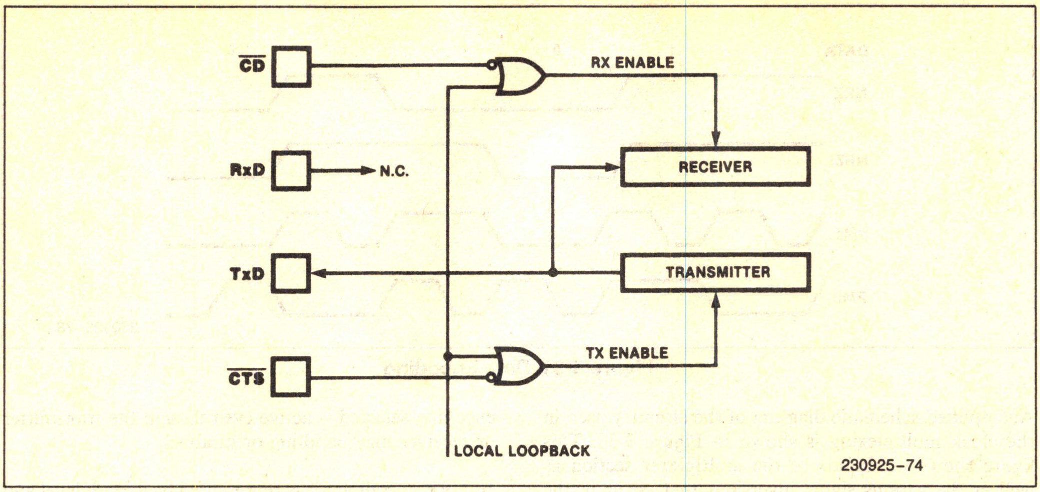


Figure 3-32. Local Loopback

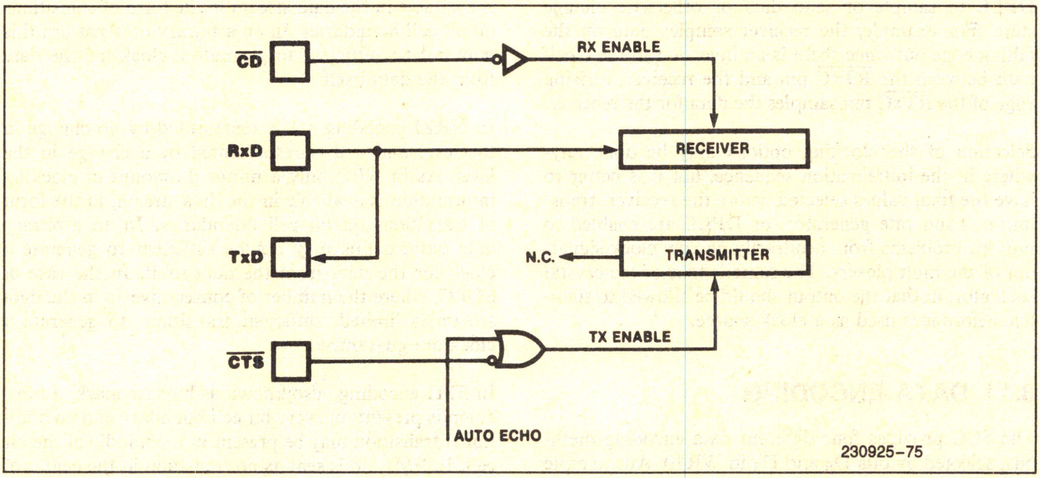


Figure 3-33. Auto Echo

In FM0 encoding, also known as biphase space, a transition is present on every bit cell boundary and an additional transition may be present in the middle of the bit cell. In FM0 a 1 is sent as no transition in the center of the bit cell and a 0 is sent as a transition in the center of the bit cell. Because of all the transitions, FM0 encoded data contains sufficient information to recover a clock from the data.

The data encoding method should be selected in the initialization procedure before the transmitter and receiver are enabled but no other restrictions apply. Note that in NRZ and NRZI the receiver samples the data only on one edge, as shown in Figure 3-30. However, in FM1 and FM0 the receiver samples the data on both edges. Also, as shown in Figure 3-30 the transmitter defines bit cell boundaries by one edge in all cases and uses the other edge in FM1 and FM0 to create the mid-bit transition.

3.12 MISCELLANEOUS FEATURES

The SCC contains two other features useful for diagnostic purposes, controlled by bits in WR14. They are local loopback and auto echo. Local loopback is selected when bit D_4 of WR14 is set to 1. In this mode the output of the transmitter is internally connected to the input of the receiver. At the same time the TxD pin remains connected to the transmitter. In this mode the $\overline{CD}$ pin is ignored as a receiver enable and the $\overline{CTS}$ pin is ignored as a transmitter enable even if the Auto Enables mode has been selected. Note that the DPLL input is connected to the RxD pin, not to the input of the receiver. This precludes the use of the DPLL in local loopback. Auto echo is selected when bit D_3 of WR14 is set to 1. In this mode the TxD pin is connected directly to the RxD pin, and the receiver input is connected to the RxD pin . In this mode the $\overline{CTS}$ pin is ignored as a transmitter enable and the output of the transmitter does not connect to anything. If both the Local Loopback and Auto Echo bits are set to 1 the auto echo mode will be selected, but both the $\overline{CTS}$ pin and $\overline{CD}$ pin will be ignored as auto enables. This should not be considered a normal operating mode, however. Local Loopback is shown schematically in Figure 3-32 and auto echo is shown schematically in Figure 3-33.

CHAPTER 4
REGISTER DESCRIPTION

4.0 REGISTER DESCRIPTION

The following sections describe the SCC registers. Each register is detailed in terms of bit configuration, the active states of each bit, their definitions, their functions, and their effects upon the internal hardware and external pins.

4.1 WRITE REGISTERS

The SCC write register set includes ten control registers, two sync character registers, two baud rate time constant registers and a transmit buffer in each channel, and one master interrupt register. The following sections describe in detail each write register and the associated bit configuration for each.

4.1.1 Write Register 0 (Command Register)

WR0 is the command register and the CRC reset code register. Figure 4-1 shows the bit configuration of WR0.

Bits D7-D6: CRC Reset Codes 0 and 1

Null Codes (00). This command has no effect on the SCC and is used when a write to WR0 is necessary for some reason other than a CRC Reset command.

Reset Receive CRC Checker (01). This command is used to initialize the receive CRC circuitry. It is necessary in synchronous modes (except SDLC) if the Enter Hunt Mode command in Write Register 3 is **not** issued between received messages. Any action that disables the receiver initializes the CRC circuitry. Resetting the Receive CRC Checker command is accomplished automatically in SDLC mode.

Reset Transmit CRC Generator (10). This command initializes the CRC generator. It is usually issued in the initialization routine and after the CRC has been transmitted. A Channel Reset will not initialize the generator and this command should not be issued until after the transmitter has been enabled in the initialization routine.

Reset Transmit Underrun/EOM Latch (11). This command controls the transmission of CRC at the end of transmission (EOM). If this latch has been reset and a transmit underrun occurs, the SCC automatically appends CRC to the message. In SDLC mode with Abort On Underrun selected, the SCC sends an abort, and

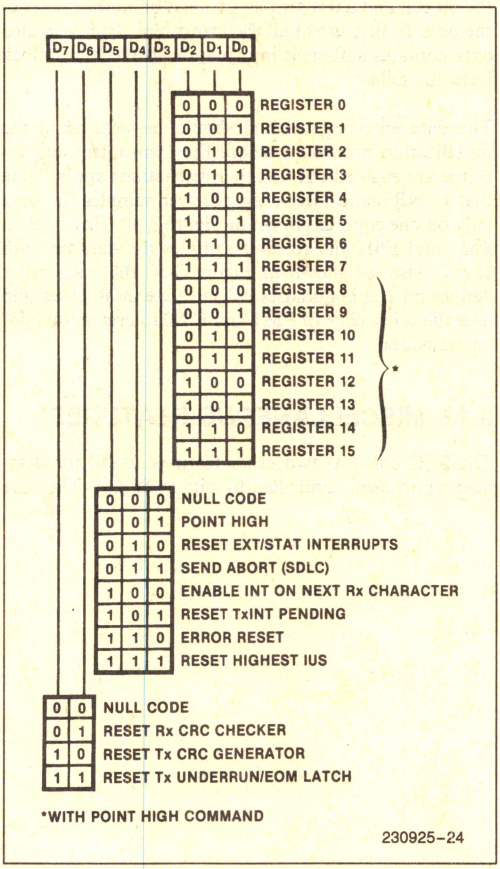


Figure 4-1. Command Register: WR0:

Flag on underrun if the Tx Underrun/EOM latch has been reset.

At the start of the CRC transmission, the Tx Underrun/EOM latch is set. The Reset command can be issued at any time during a message. If the transmitter is disabled, this command will not reset the latch. However, if no External Status interrupt is pending or if a Reset External Status Int command accompanies this command while the transmitter is disabled, an External/Status interrupt is generated with the Tx Underrun/EOM bit rest in RR0.

Bits D5-D3: Command Codes

Null Code (000). The Null command has no effect on the SCC.

Point High (001). This command effectively adds eight to the Register Pointer (D2-D0) by allowing WR8 through WR15 to be accessed. The Point High command and the Register Pointer bits are written simultaneously.

Reset External/Status Interrupts (010). After an External/Status interrupt (a change on a modem line or a break condition, for example), the status bits in RR0 are latched. This command re-enables the bits and allows interrupts to occur again as a result of a status change. Latching the status bits captures short pulses until the CPU has time to read the change. The SCC contains simple queueing logic associated with most of the external status bits in RR0. If another External/Status condition changes while a previous condition is still pending (Reset External/Status Interrupts has not yet been issued) and this condition persists until after the command is issued, this second change causes another External/Status interrupt. However, if this second status change does not persist (there are two transitions), another interrupt is not generated. Exceptions to this rule are detailed in the RR0 description.

Send Abort (011). This command is used in SDLC mode to transmit a sequence of eight to thirteen 1s. This command always empties the transmit buffer and sets the Tx Underrun/EOM bit in Read Register 0.

Enable Interrupt on Next Rx Character (100). If the interrupt on the First Received Character mode is selected, this command is used to reactivate that mode after each message is received. The next character to enter the receive FIFO causes a Receive interrupt. Alternatively, the first previously stored character in the FIFO will cause a Receive interrupt.

Reset Tx Interrupt Pending (101). This command is used in cases where there are no more characters to be sent; e.g., at the end of a message. This command prevents further transmit interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent. This command is necessary to prevent the transmitter from requesting an interrupt when the transmit buffer becomes empty (with Transmit Interrupt Enabled).

Error Reset (110). This command resets the error bits in RR1. If Interrupt on First Rx Character or Interrupt on Special Condition modes are selected and a special condition exists, the data with the special condition is held in the receive FIFO until this command is issued. If either of these modes is selected and this command is issued before the data has been read from the receive FIFO, the data is lost.

Reset Highest IUS (111). This command resets the highest priority Interrupt Under Service (IUS) bit, allowing lower priority conditions to request interrupts. This command allows the use of the internal daisy chain (even in systems without an external daisy chain) and should be the last operation in an interrupt service routine.

Bits 2 through 0: Register Selection Code

These three bits select Registers 0 through 7. With the Point High command, Registers 8 through 15 are selected.

The following is a summary of the bit descriptions for each write register (WR1–WR15).

4.1.2 Write Register 1 (Transmit/Receive Interrupt and Data Transfer Mode Definition)

Write Register 1 is the control register for the various SCC interrupt and Ready/Request modes. Figure 4-2 shows the bit assignments for WR1.

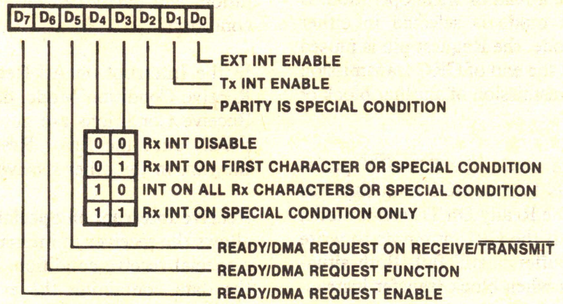


Figure 4-2. Write Register 1

Bit 7: READY/DMA Request Enable

This bit enables the $\overline{\text{READY/REQUEST}}$ function in conjunction with the $\overline{\text{READY/REQUEST}}$ Function Select bit (D6). If bit 7 is set to 1, the state of bit 6 determines the activity of the $\overline{\text{READY/REQUEST}}$ pin (Ready or Request). If bit 7 is set to 0, the selected function (bit 6) forces the $\overline{\text{READY/REQUEST}}$ pin into the appropriate inactive state (High for Request, floating for Ready).

Bit 6: READY/DMA Request Function

The request function is selected by setting this bit to 1. In the Request mode, the $\overline{\text{READY/REQUEST}}$ pin switches from High to Low when the SCC is ready to transfer data. When this bit is 0, the Ready function is selected. In the Ready mode, the $\overline{\text{READY/REQUEST}}$ pin switches from floating to Low when the CPU attempts to transfer data before the SCC is ready.

Bit 5: READY/DMA Request on Receive Transmit

This bit determines whether the $\overline{\text{READY/REQUEST}}$ pin operates in the Transmit mode or the Receive mode. When set to 1, this bit allows the $\overline{\text{READY/REQUEST}}$ function to follow the state of the receive buffer; i.e., depending on the state of bit 6, the $\overline{\text{READY/REQUEST}}$ pin is active or inactive in relation to the empty or full state of the receive buffer. Conversely, if this bit is set to 0, the state of the $\overline{\text{READY/REQUEST}}$ pin is determined by bit 6 and the state of the transmit buffer. (Note that a transmit request function is available on the $\overline{\text{DTR/REQ}}$ pin. This allows full-duplex operation under DMA control for both channels).

The request function may occur only when the SCC is **not** selected; e.g., if the internal request becomes active while the SCC is in the middle of a read or write cycle, the external request will not become active until the cycle is complete. An active request output causes a DMA controller to initiate a read or write operation. If the request on Transmit mode is selected in either SDLC or Synchronous Mode, the Request pin is pulsed Low for one CLK cycle at the end of CRC transmission to allow the immediate transmission of another block of data.

In the Ready On Receive mode, the $\overline{\text{READY}}$ pin is active if the CPU attempts to read SCC data that has not yet been received. In the Ready On Transmit mode, the $\overline{\text{READY}}$ pin is active if the CPU attempts to write data when the transmit buffer is still full. Both situations can occur frequently when block transfer instructions are used.

Bits 4 and 3: Receive Interrupt Modes

These two bits specify the various character-available conditions that may cause interrupt requests.

Receiver Interrupts Disabled (00). This mode prevents the receiver from requesting an interrupt and is normally used in a polled environment where either the status bits in RR0 or the modified vector in RR2 (Channel B) can be monitored to initiate a service routine. Although the receiver interrupts are disabled, a special condition can still provide a unique vector status in RR2.

Receive Interrupt on First Character or Special Condition (01). The receiver requests an interrupt in this mode on the first available character (or stored FIFO character) or on a special condition. Sync characters to be stripped from the message stream do not cause interrupts.

Special receive conditions are: receiver overrun, framing error, end of frame, or parity error (if selected). If a special receive condition occurs, the data containing the error is stored in the receive FIFO until an Error Reset command is issued by the CPU.

This mode is usually selected when a Block Transfer mode is used. In this interrupt mode, a pending special receive condition remains set until either an Error Reset command, a channel or hardware reset, or until receive interrupts are disabled.

The Receive Interrupt on First Character or Special Condition mode can be re-enabled by the Enable Rx Interrupt on Next Character command in WR0.

Interrupt on All Receive Characters or Special Condition (10). This mode allows an interrupt for every character received (or character in the receive FIFO) and provides a unique vector when a special condition exists. The Receiver Overrun bit and the Parity Error bit in RR1 are two special conditions that are latched. These two bits must be reset by the Error Reset command. Receiver overrun is always a special receive condition, and parity can be programmed to be a special condition.

In the Interrupt on All Receive Characters or Special Receive Condition Mode, data characters with Special Receive Conditions are not held locked in the receive FIFO, until the Error Reset Command is issued, as they are in the other receive interrupt modes.

Receive Interrupt on Special Condition (11). This mode allows the receiver to interrupt only on characters with a special receive condition. When an interrupt occurs, the data containing the error is held in the receive FIFO until an Error Reset command is issued. When using this mode in conjunction with a DMA, the DMA can be initialized and enabled before any characters

have been received by the SCC. This eliminates the time-critical section of code required in the Receive Interrupt on First Character or Special condition mode; i.e., all data can be transferred via the DMA so that the CPU needs not handle the first received character as a special case.

Bit 2: Parity Is Special Condition

If this bit is set to 1, any received characters with parity not matching the sense programmed in WR4 give rise to a Special Receive Condition. If parity is disabled (WR4), this bit is ignored. A special condition modifies the status of the interrupt vector stored in WR2. During an interrupt ackowledge cycle, this vector can be placed on the data bus.

Bit 1: Transmitter Interrupt Enable

If this bit is set to 1, the transmitter requests an interrupt whenever the transmit buffer becomes empty.

Bit 0: External/Status Master Interrupt Enable

This bit is the master enable for External/Status interrupts including $\overline{CD}$, $\overline{CTS}$, $\overline{SYNC}$ pins, break, abort, the beginning of CRC transmission when the Transmit/Underrun/EOM latch is set, or when the counter in the baud rate generator reaches 0. Write Register 15 contains the individual enable bits for each of these sources of External/Status interrupts. This bit is reset by a channel or hardware reset.

4.1.3 Write Register 2 (Interrupt Vector)

WR2 is the interrupt vector register. Only one register vector exists in the SCC, but it can be accessed through either channel. The interrupt vector can be modified by status information. This is controlled by the Vector Includes Status (VIS) and the Status High/Status Low bits in register WR9. The bit positions for WR2 are shown in Figure 4-3.

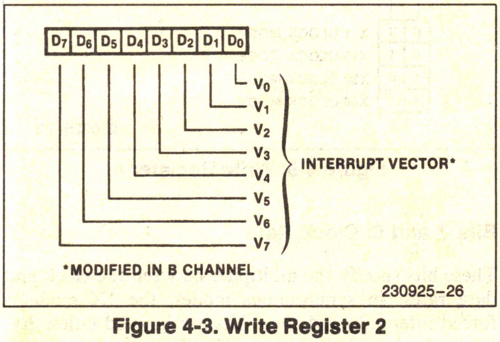


Figure 4-3. Write Register 2

4.1.4 Write Register 3 (Receive Parameters and Control)

This register contains the control bits and parameters for the receiver logic illustrated in Figure 4-4.

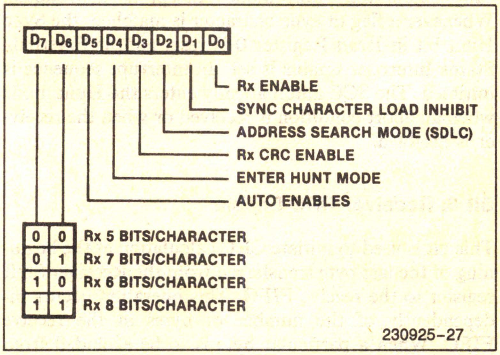


Figure 4-4. Write Register 3

Bits 7 and 6: Receiver Bits/Character

The state of these two bits determines the number of bits to be assembled as a character in the received serial data stream. The number of bits per character can be changed while a character is being assembled but only **before** the number of bits currently programmed is reached. Unused bits in the Received Data Register (RR8) are set to 1 in asynchronous modes. In synchronous modes and SDLC modes, the SCC merely transfers an 8-bit section of the serial data stream to the receive of the serial data stream to the receive FIFO at the appropriate time. Table 4-1 lists the number of bits per character in the assembled character format.

Table 4-1. Receive Bits/Character

D_7	D_6	
0	0	5 Bits/Character
0	1	7 Bits/Character
1	0	6 Bits/Character
1	1	8 Bits/Character

Bit 5: Auto Enables

This bit programs the function for both the $\overline{CD}$ and $\overline{CTS}$ pins. $\overline{CTS}$ becomes the transmitter enable and $\overline{CD}$ becomes the receiver enable when this bit is set to 1. However, the Receiver Enable and Transmit Enable bits must be set before the $\overline{CD}$ and $\overline{CTS}$ pins can be used in this manner. When the Auto Enables bit is set to 0, the $\overline{CD}$ and $\overline{CTS}$ pins are merely inputs to the corresponding status bits in Read Register 0. The state of $\overline{CD}$ is ignored in the Local Loopback mode. The state of $\overline{CTS}$ is ignored in both Auto Echo and Local Loopback modes.

Bit 4: Enter Hunt Mode

This command forces the comparison of sync characters or flags to assembled receive characters for the purpose of synchronization. After reset, the SCC automatically enters the Hunt mode (except asynchronous). Whenever a flag or sync character is matched, the Sync Hunt bit in Read Register 0 is reset and, if External/Status Interrupt Enable is set, an interrupt sequence is initiated. The SCC automatically enters the Hunt mode when an abort condition is received or when the receiver is disabled.

Bit 3: Receiver CRC Enable

This bit is used to initiate CRC calculation at the beginning of the last byte transferred from the Receiver Shift register to the receive FIFO. This operation occurs independently of the number of bytes in the receive FIFO. When a particular byte is to be excluded from CRC calculation, this bit should be reset before the next byte is transferred to the receive FIFO.

This bit is internally set to 1 in SDLC mode and the SCC calculates CRC on all bits except between the opening and closing character flags. This bit is ignored in asynchronous modes. If this feature is used, care must be taken to ensure that eight bits per character is selected in the receiver because of an inherent delay from the Receive Shift register to the CRC checker.

Bit 2: Address Search Mode (SDLC)

Setting this bit in SDLC mode causes messages with addresses not matching the address programmed in WR6 to be rejected. No receiver interrupts can occur in this mode unless there is an address match. The address that the SCC attempts to match can be unique (1 in 256) or multiple (16 in 256), depending on the state of Sync Character Load Inhibit bit. The Address Search mode bit is ignored in all modes except SDLC.

Bit 1: Synch Character Load Inhibit

If this bit is set to 1 in any synchronous mode except SDLC, the SCC compares the byte in WR6 with the byte about to be stored in the FIFO, and it inhibits this load if the bytes are equal. The SCC does not calculate the CRC on bytes stripped from the data stream in this manner. If the 6-bit sync option is selected while in Monosync mode, the compare is still across eight-bits, so WR6 must be programmed for proper operation.

If the 6-bit sync option is selected with this bit set to 1, all sync characters except the one immediately preceding the data are stripped from the message. If the 6-bit sync option is selected while in the Bisync mode, this bit is ignored.

The address recognition logic of the receiver is modified in SDLC mode if this bit is set to 1; i.e., only the four most significant bits of WR6 must match the receiver address. This procedure allows the SCC to receive frames from up to 16 separate sources without programming WR6 for each source (if each station address has the four most significant bits in common). The address field in the frame is still eight bits long.

This bit is ignored in SDLC mode if Address Search mode has not been selected.

Bit 0: Receiver Enable

When this bit is set to 1, receiver operation begins. This bit should be set only after all other receiver parameters are established and the receiver is completely initialized. This bit is reset by a channel or hardware reset command, and it disables the receiver.

4.1.5 Write Register 4 (Transmit/Receiver Miscellaneous Parameters and Modes)

WR4 contains the control bits for both the receiver and the transmitter. These bits should be set in the transmit and receiver initialization routine before issuing the contents of WR1, WR3, WR6, and WR7. Bit positions for WR4 are shown in Figure 4-5.

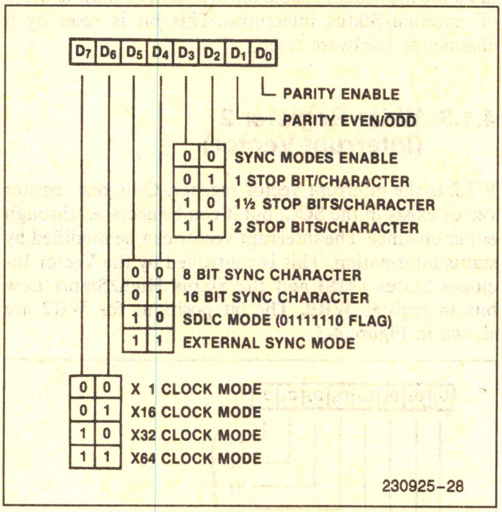


Figure 4-5. Write Register 4

Bits 7 and 6: Clock Rate

These bits specify the multiplier between the clock and data rates. In synchronous modes, the 1X mode is forced internally and these bits are ignored unless External Sync mode has been selected.

1X Mode (00). The clock rate and data rate are the same. In External Sync mode, this bit combination specifies that only the $\overline{SYNC}$ pin can be used to achieve character synchronization.

16X Mode (01). The clock rate is 16 times the data rate. In External Sync mode, this bit combination specifies that only the $\overline{SYNC}$ pin can be used to achieve character synchronization.

32X Mode (10). The clock rate is 32 times the data rate. In External Sync mode, this bit combination specifies that either the $\overline{SYNC}$ pin or a match with the character stored in WR7 will signal character synchronization. The sync character can be either six or eight bits long as specified by the 6-bit/8-bit Sync bit in WR10.

64X Mode (11). The clock rate is 64 times the data rate. With this bit combination in External Sync mode, both the receiver and transmitter are placed in SDLC mode. The only variation from normal SDLC operation is that the $\overline{SYNC}$ pin can be used to start or stop the reception of a frame by forcing the receiver to act as though a flag had been received.

Bits 5 and 4: Sync Modes

These two bits select the various options for character synchronization. They are ignored unless synchronization modes are selected in the stop bits field of this register.

Monosync (00). In this mode, the receiver achieves character synchronization by matching the character stored in WR7 with an identical character in the received data stream. The transmitter uses the character stored in R6 as a time fill. The sync character can be either six or eight bits, depending on the state of the 6-bit/8-bit Sync bit in WR10. If the Sync Character Load Inhibit bit is set, the receiver strips the contents of WR6 from the data stream if received within character boundaries.

Bisync (01). The concatenation of WR7 with WR6 is used for receiver synchronization and as a time fill by the transmitter. The sync character can be 12 or 16 bits in the receiver, depending on the state of the 6-bit/8-bit Sync bit in WR10. The transmitted character is always 16 bits.

SDLC Mode (10). In this mode, SDLC is selected and requires a Flag (01111110) to be written to WR7. The receiver address field should be written to WR6. The SDLC CRC polynomial must also be selected (WR5) in SDLC mode.

External Sync Mode (11). In this mode, the SCC expects external logic to signal character synchronization via the $\overline{SYNC}$ pin. If the crystal oscillator option is selected (in WR11), the internal $\overline{SYNC}$ signal is forced to 0. In this mode, bits D7–D6 of this register select special versions of External Sync mode. In this mode, the transmitter is in Monosync mode using the contents of WR6 as the time fill with the sync character length specified by the 6-bit/8-bit Sync bit in WR10.

Bits 3 and 2: Stop Bits

These bits determine the number of stop bits added to each asynchronous character that is transmitted. The receiver always checks for one stop bit in Asynchronous mode. A Special mode specifies that a Synchronous mode is to be selected. D2 is always set to 1 by a channel or hardware reset to ensure that the $\overline{SYNC}$ pin is in a known state after a reset.

Synchronous Modes Enable (00). This bit combination selects one of the synchronous modes specified by bits D4, D5, D6, and D7 of this register and forces the 1X Clock mode internally.

1 Stop Bits/Character (01). This bit selects Asynchronous mode with one stop bit per character.

1½ Stop Bits/Character (10). These bits select Asynchronous mode with 1½ stop bits per character. **This mode can not be used with the 1X clock mode.**

2 Stop Bits/Character (11). These bits select Asynchronous mode with two stop bits per transmitted character and check for one received stop bit.

Bit 1: Parity Even/$\overline{\text{Odd}}$

This bit determines whether parity is checked as even or odd. A 1 programmed here selects even parity, and a 0 selects odd parity. This bit is ignored if the Parity bit is not set.

Bit 0: Parity Enable

When this bit is set, an additional bit position beyond those specified in the bits/character control is added to the transmitted data and is expected in the receive data. The Received Parity bit is transferred to the CPU as part of the data unless eight bits per character is selected in the receiver.

4.1.6 Write Register 5 (Transmit Parameter and Controls)

WR5 contains control bits that affect the operation of the transmitter. D2 affects both the transmitter and the receiver. Bit positions for WR5 are shown in Figure 4-6.

Bit 7: Data Terminal Ready

This is the control bit for the $\overline{\text{DTR/REQ}}$ pin while the pin is in the $\overline{\text{DTR}}$ mode (selected in WR14). When set, $\overline{\text{DTR}}$ is Low; when reset, $\overline{\text{DTR}}$ is High. This bit is ignored when $\overline{\text{DTR/REQ}}$ is programmed to act as a $\overline{\text{REQUEST}}$ pin. This bit is reset by a channel or hardware reset.

Bits 6 and 5: Tx Bits/Character 1 and 0

These bits control the number of bits in each byte transferred to the transmit buffer. Bits sent must be right justified with least significant bits first.

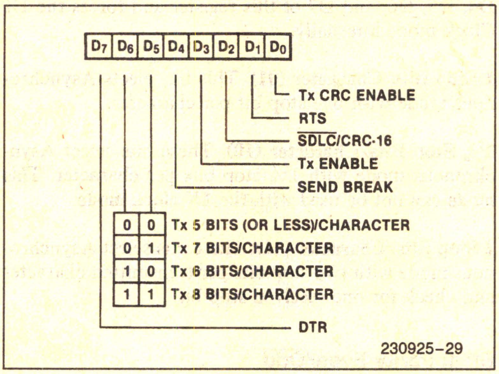


Figure 4-6. Write Register 5

The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown below in Table 4.2. In the Six or Seven Bits/Character modes, unused data bits are ignored.

Table 4-2. Tx Bits/Character 1 and 0

Tx Bits/ Char 1	Tx Bits/ Char 0							
0	0	5 or less bits/character						
0	1	7 bits/character						
1	0	6 bits/character						
1	1	8 bits/character						
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	0	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
0	0	0	D	D	D	D	D	Sends five data bits

Bit 4: Send Break

When set, this bit forces the TxD output to send continuous 0s beginning with the following transmit clock, regardless of any data being transmitted at the time. This bit functions whether or not the transmitter is enabled. When reset, TxD continues to send the contents of the Transmit Shift register, which might be syncs, data, or all 1s. If this bit is set while in the X21 mode (Monosync and Loop mode selected) and character synchronization is achieved in the receiver, this bit is automatically reset and the transmitter begins sending syncs or data. This bit can also be reset by a channel or hardware reset.

Bit 3: Transmit Enable

Data is not transmitted until this bit is set, and the TxD output sends continuous 1s unless Auto Echo mode or SDLC Loop mode is selected. If this bit is reset after transmission started, the transmission of data or sync characters is completed. If the transmitter is disabled during the transmission of a CRC character, sync or flag characters are sent instead of CRC. This bit is reset by a channel or hardware reset.

Bit 2: $\overline{\text{SDLC}}$/CRC-16

This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial is used; when reset, the SDLC polynomial is used. The SDLC/CRC polynomial must be selected when SDLC mode is selected. The CRC generator and checker can be preset to all 0s all 1s, depending on the state of the Preset 1/Preset 0 bit in WR10.

Bit 1: Request to Send

This is the control bit for the $\overline{\text{RTS}}$ pin. When the RTS bit is set, the $\overline{\text{RTS}}$ pin goes Low; when reset, $\overline{\text{RTS}}$ goes High. In the Asynchronous mode with the Auto Enables bit set, $\overline{\text{RTS}}$ goes High only after all bits of the character have been sent and the transmit buffer is empty. In synchronous modes or the Asynchronous mode with auto enables off, the pin directly follows the state of this bit. This bit is reset by a channel or hardware reset.

Bit 0: Transmit CRC Enable

This bit determines whether or not CRC is calculated on a transmit character. If this bit is set at the time the character is loaded from the transmit buffer to the Transmit Shift register, CRC is calculated on that character. CRC is not automatically sent unless this bit is set when the transmit underrun exists.

D7	D6	D5	D4	D3	D2	D1	D0	
SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	MONOSYNC, 8 BITS
SYNC$_1$	SYNC$_0$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	MONOSYNC, 6 BITS
SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	BISYNC, 16 BITS
SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	1	1	1	1	BISYNC, 12 BITS
ADR$_7$	ADR$_6$	ADR$_5$	ADR$_4$	ADR$_3$	ADR$_2$	ADR$_1$	ADR$_0$	SDLC
ADR$_7$	ADR$_6$	ADR$_5$	ADR$_4$	x	x	x	x	SDLC (ADDRESS RANGE)

230925-31

Figure 4-7. Write Register 6

D7	D6	D5	D4	D3	D2	D1	D0	
SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	MONOSYNC, 8 BITS
SYNC$_5$	SYNC$_4$	SYNC$_3$	SYNC$_2$	SYNC$_1$	SYNC$_0$	x	x	MONOSYNC, 6 BITS
SYNC$_{15}$	SYNC$_{14}$	SYNC$_{13}$	SYNC$_{12}$	SYNC$_{11}$	SYNC$_{10}$	SYNC$_9$	SYNC$_8$	BISYNC, 16 BITS
SYNC$_{11}$	SYNC$_{10}$	SYNC$_9$	SYNC$_8$	SYNC$_7$	SYNC$_6$	SYNC$_5$	SYNC$_4$	BISYNC, 12 BITS
0	1	1	1	1	1	1	0	SDLC

230925-30

Figure 4-8. Write Register 7

4.1.7 Write Register 6 (Sync Characters or SDLC Address Field)

WR6 is programmed to contain the transmit sync character in the Monosync mode, the first byte of a 16-bit sync character in the External Sync mode. WR6 is not used in asynchronous modes. In the SDLC modes, it is programmed to contain the secondary address field used to compare against the address field of the SDLC Frame. In SDLC mode, the SCC does not automatically transmit the station address at the beginning of a response frame. WR6 is not used in asynchronous mode. Bit positions for WR6 are shown in Figure 4-7.

4.1.8 Write Register 7 (Sync Character or SDLC Flag)

WR7 is programmed to contain the receive sync character in the Monosync mode, a second byte (the last eight bits) of a 16-bit sync character in the Bisync mode, or a Flag character (01111110) in the SDLC modes. WR7 may hold the receive sync character or a flag if one of the special versions of the External Sync mode is selected. WR7 is not used in Asynchronous mode. Bit positions for WR7 are shown in Figure 4-8.

4.1.9 Write Register 8 (Transmit Buffer)

WR8 is the transmit buffer register.

4.1.10 Write Register 9 (Master Interrupt Control)

WR9 is the Master Interrupt Control register and contains the Reset command bits. Only one WR9 exists in the SCC and can be accessed from either channel. The interrupt control bits can be programmed at the same time as the Reset command because these bits are only reset by a hardware reset. Bit positions for WR9 are shown in Figure 4-9.

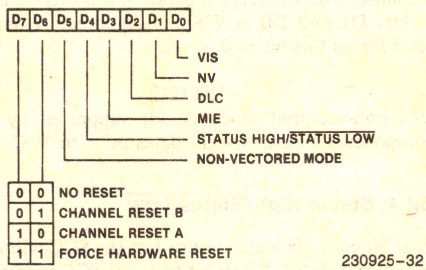


Figure 4-9. Write Register 9

Bit 7-6: Reset Command Bits

Together, these bits select one of the reset commands for the SCC. Setting either of these bits to 1 disables both the receiver and the transmitter in the corresponding channel, forces TxD for that channel marking, forces the modem control signals High in that channel, resets all IPs and IUSs and disables all interrupts in that channel. Four extra CLK cycles must be allowed beyond the usual cycle time after any of the active reset commands are issued before any additional commands or controls are written to the channel affected. Four extra CLK cycles must be allowed beyond the usual cycle time before any additional command or controls are written to the SCC.

No Reset (00). This command has no effect. It is used when a write to WR9 is necessary for some reason other than an SCC Reset command.

Channel Reset B (01). Issuing this command causes a channel reset to be performed on Channel B.

Channel Reset A (10). Issuing this command causes a channel reset to be performed on Channel A.

Force Hardware Reset (11). The effects of this command are identical to those of a hardware reset, except that the MIE, Status High/Status Low and DLC bits take the programmed values that accompany this command.

Bit 5: Non-Vectored Mode*

This bit selects the method used to acknowledge an 82530 interrupt request. If this bit is set to 0 (Vectored Mode), the 82530 will expect an Interrupt Acknowledge cycle in response to an Interrupt Request (see section 3.3.3). In the Vectored Mode, reading RR2 provides the interrupt vector without any effect on the internal interrupt logic. If this bit is set to 1 (Non-Vectored Mode), the 82530 will expect interrupt acknowledgement through the reading of RR2 (see section 3.3.4). In the Non-Vectored Mode, $\overline{\text{INTA}}$ must be tied inactive high, and a read to RR2 should only occur in response to an interrupt request. When this bit is set to 1, bits D1 and D0 in WR9 are ignored. A hardware reset forces this bit to 0.

*NOTE:
The non-vectored mode is not supported by 82530 components with copyright dates prior to 1986.

Bit 4: Status High/$\overline{\text{Status Low}}$

This bit controls which vector bits the SCC will modify to indicate status. When set to 1, the SCC modifies bits V6, V5, and V4 according to Table 4-3. When set to 0, the SCC modifies bits V1, V2 and V3 according to Table 4-3. This bit controls status in both the vector returned during an interrupt acknowledge cycle and the status in RR2. This bit is reset by a hardware reset.

Bit 3: Master Interrupt Enable

The Master Interrupt Enable bit is used to globally inhibit SCC interrupts. Setting this bit to 0 has the same effect on the internal interrupt logic as does a low on the IEI pin, except that IEO is not forced Low. When this bit is set to 0, IEO is not forced Low. When this bit is set to 0, IEO follows the state of IEI unless there is an IUS set in the SCC. No IUS can be set after the MIE bit is set to 0. This bit is reset by a hardware reset.

Bit 2: Disable Lower Chain

The Disable Lower Chain bit can be used by the CPU to control the interrupt daisy chain. Setting this bit to 1 forces the IEO pin Low, preventing lower-priority devices on the daisy chain from requesting interrupts. This bit is reset by a hardware reset.

Bit 1: No Vector (Vectored Mode Only)

The No Vector bit controls whether or not the SCC will respond to an interrupt acknowledge cycle by placing a vector on the data bus if the SCC is the highest-priority device requesting an interrupt. If this bit is set, no vector is returned; i.e., AD_0-AD_7 remain 3-stated during an interrupt acknowledge cycle, even if the SCC is the highest-priority device requesting an interrupt.

Bit 0: Vector Includes Status (Vectored Mode Only)

The Vector Includes status bit controls whether or not the SCC will include status information in the vector it places on the bus in response to an interrupt acknowledge cycle. If this bit is set, the vector returned is variable, with the variable field depending on the highest-priority IP that is set. Table 4-3 shows the encoding of the status information. This bit is ignored if the No Vector (NV) bit is set.

Table 4-3. Status Vector Mode Table

V_3	V_2	V_1	Status High/Status Low = 0
V_4	V_5	V_6	Status High/Status Low = 1
0	0	0	Ch B transmit buffer Empty
0	0	1	Ch B External/Status Change
0	1	0	Ch B Receive Character Available
0	1	1	Ch B Special Receive Condition
1	0	0	Ch A transmit buffer Empty
1	0	1	Ch A External/Status Change
1	1	0	Ch A Receive Character Available
1	1	1	Ch A Special Receive Condition

4.1.11 Write Register 10 (Miscellaneous Transmitter/ Receiver Control Bits)

WR10 contains miscellaneous control bits for both the receiver and the transmitter. Bit positions for WR10 are shown in Figure 4-10.

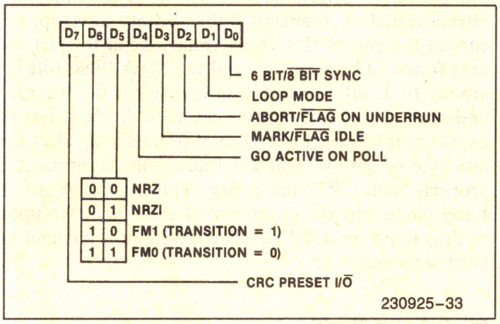


Figure 4-10. Write Register 10

Bit 7: CRC Preset I/$\overline{\text{O}}$

This bit specifies the initialized condition of the receive CRC checker and the transmit CRC generator. If this bit is set to 1, the CRC generator and checker are preset to 1. If this bit is set to 0, the CRC generator and checker are preset to 0. Either option can be selected with either CRC polynomial. In SDLC mode, the transmitted CRC is inverted before transmission and the received CRC is checked against the bit pattern 0001110100001111. This bit is reset by a channel or hardware reset. This bit is ignored in Asynchronous mode.

Bits 6 and 5: Data Encoding 1 and 2

These bits control the coding method used for both the transmitter and the receiver, as illustrated in Table 4-4. All of the clocking options are available for all coding methods. The DPLL in the SCC is useful for recovering clocking information in NRZI and FM modes. Any coding method can be used in any 1x mode. A hardware reset forces NRZ mode. Timing for the various modes is shown in Figure 4-11.

Table 4-4. Data Encoding

Data	Data Encoding	Encoding
0	0	NRZ
0	1	NRZI
1	0	FM1 (transition = 1)
1	1	FM0 (transition = 1)

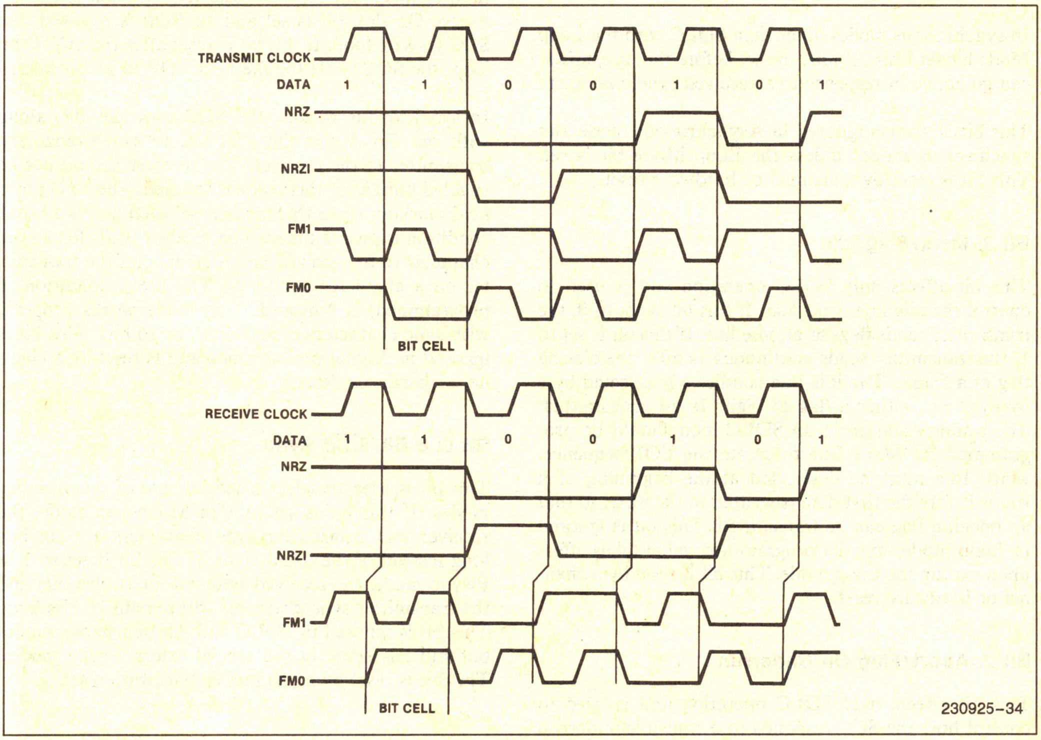


Figure 4-11. NRZ(NRZI)/FM1(FM0) Timing

Bit 4: Go Active On Poll

When Loop mode is first selected during SDLC operation, the SCC connects RxD to TxD with only gate delays in the path. The SCC does not go on-loop and insert the 1-bit delay between RxD and TxD until this bit has been set and an EOP received. When the SCC is on-loop, the transmitter cannot go active unless this bit is set at the time an EOP is received. The SCC examines this bit whenever the transmitter is active in SDLC Loop mode and is sending a flag. If this bit is set at the time the flag is leaving the Transmit Shift register, another flag or data byte (if the transmit buffer is full) is transmitted. If the Go Active on Poll bit is not set at this time, the transmitter finishes sending the flag and reverts to the 1-Bit Delay mode. Thus, to transmit only one response frame, this bit should be reset after the first data byte is sent to the SCC but before CRC has been transmitted. If the bit is not reset before CRC is transmitted, extra flags are sent, slowing down response time on the loop. If this bit is reset before the first data is written, the SCC completes the transmission of the present flag and reverts to the 1-Bit Delay mode. After gaining control of the loop, the SCC is not able to transmit again until a flag and another EOP have been received. Though not strictly necessary, it is good practice to set this bit only upon receipt of a poll frame to ensure that the SCC does not go on loop without the CPU noticing it.

In synchronous modes other than SDLC with the Loop Mode bit set, this bit must be set before the transmitter can go active in response to a received sync character.

This bit is always ignored in Asynchronous mode and synchronous modes unless the Loop Mode bit is set. This bit is reset by a channel or hardware reset.

Bit 3: Mark/$\overline{\text{Flag}}$ Idle

This bit affects only SDLC operation and is used to control the idle line condition. If this bit is set to 0, the transmitter sends flags as an idle line. If this bit is set to 1, the transmitter sends continuous 1s after the closing flag of a frame. The idle line condition is selected byte by byte; i.e., either a flag or eight 1s are transmitted. The primary station in an SDLC loop should be programmed for Mark Idle to create the EOP sequence. Mark Idle must be deselected at the beginning of a frame before the first data is written to the SCC, so that an opening flag can be transmitted. This bit is ignored in Loop mode, but the programmed value takes effect upon exiting the Loop mode. This bit is reset by a channel or hardware reset.

Bit 2: Abort/$\overline{\text{Flag}}$ On Underrun

This bit affects only SDLC operation and is used to control how the SCC responds to a transmit underrun condition. If this bit is set to 1 and a transmit underrun occurs, the SCC sends an abort and a flag instead of CRC. If this bit is reset, the SCC sends CRC on a transmit underrun. At the beginning of this 16-bit transmission, the Transmit Underrun/EOM bit is set, causing an External/Status interrupt. The CPU uses this status, along with the byte count from memory or the DMA, to determine whether the frame must be retransmitted. A Transmit Buffer Empty interrupt occurs at the end of this 16-bit transmission to start the next frame. If both this bit and the Mark/$\overline{\text{Flag}}$ Idle bit are set to 1, all 1s are transmitted after the transmit underrun. This bit should be set after the first byte of data is sent to the SCC and reset immediately after the last byte of data so that the frame will be terminated properly with CRC and a flag. This bit is ignored in Loop mode, but the programmed value is active upon exiting Loop mode. This bit is reset by a channel or hardware reset.

Bit 1: Loop Mode

In SDLC mode, the initial set condition of this bit forces the SCC to connect TxD to RxD and to begin searching the incoming data stream so that it can go on loop. All bits pertinent to SDLC mode operation in other register must be set before this mode is selected. The transmitter and receiver should not be enabled until after this mode has been selected. As soon as the Go Active On Poll bit is set and an EOP is received, the SCC goes on loop. If this bit is reset after the SCC is on loop, the SCC waits for the next EOP to go off loop.

In synchronous modes, the SCC uses this bit, along with the Go Active On Poll bit, to synchronize the transmitter to the receiver. The receiver should not be enabled until after this mode is selected. The TxD pin is held marking when this mode is selected unless a break condition is programmed. The receiver waits for a sync character to be received and then enables the transmitter on a character boundary. The break condition, if programmed, is removed. This mode works properly with sync characters of six, eight, or 16 bits. This bit is ignored in Asynchronous mode and is reset by a channel or hardware reset.

Bit 0: 6 Bit/$\overline{\text{8 Bit}}$ Sync

This bit is used to select a special case of synchronous modes. If this bit is set to 1 in Monosync mode, the receiver and transmitter sync characters are six bits long instead of the usual eight. If this bit is set to 1 in Bisync mode, the received sync will be twelve bits and the transmitter sync character will remain 16 bits long. This bit is ignored in SDLC and Asynchronous modes but still has effect in the special external sync modes. This bit is reset by a channel or hardware reset.

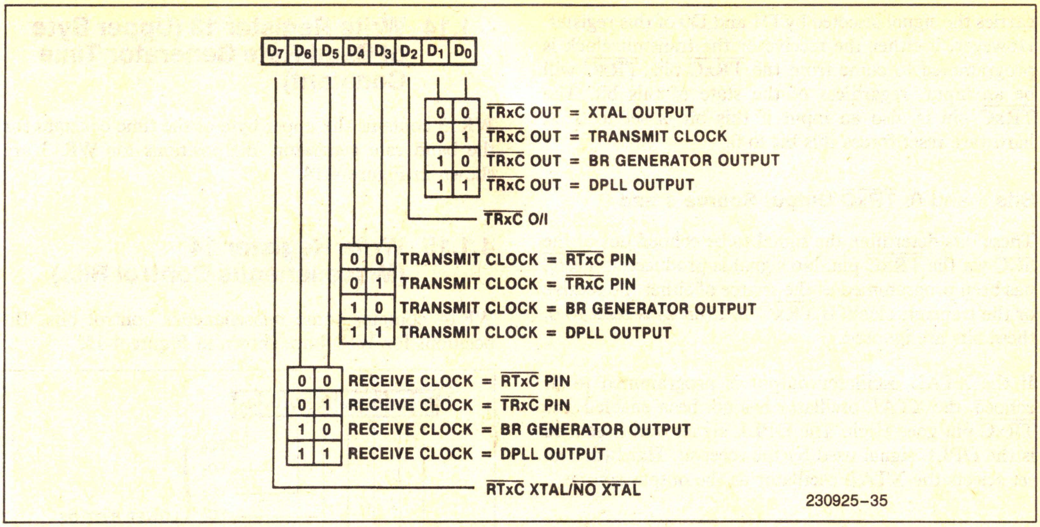


Figure 4-12. Write Register 11

4.1.12 Write Register 11 (Clock Mode Control)

WR11 is the Clock Mode Control register. The bits in this register control the sources of both the receive and transmit clocks, the type of signal on the SYNC and RTxC pins, and the direction of the TRxC pin. Bit positions for WR11 are shown in Figure 4-12.

Bit 7: RTxC XTAL/NO XTAL

This bit controls the type of input signal the SCC expects to see on the RTxC pin. If this bit is set to 0, the SCC expects a TTL-compatible signal as an input to this pin. If this bit is set to 1, the SCC connects a high-gain amplifier between the RTxC and SYNC pins in expectation of a quartz crystal being placed across the pins.

The output of this oscillator is available for use as a clocking source. In this mode of operation, the SYNC pin is unavailable for other use. The SYNC signal is forced to 0 internally. A hardware reset forces NO XTAL. (At least 20ms should be allowed after this bit is set to allow the oscillator to stabilize.)

Bit 6-5: Receive Clock 1 and 0

These bits determine the source of the receive clock as shown in Table 4-5. They do not interfere with any of the modes of operation in the SCC but simply control a multiplexer just before the internal receive clock input. A hardware reset forces the receive clock to come from the RTxC pin.

Table 4-5. Receive Clock Source

Receive Clock 1	Receive Clock 0	
0	0	Receive Clock = RTxC pin
0	1	Receive Clock = TRxC pin
1	0	Receive Clock = BR output
1	1	Receive Clock = DPLL output

Bit 4-3: Transmit Clock 1 and 0

These bits determine the source of the transmit clock as shown in Table 4-6. They do not interfere with any of the modes of operation of the SCC but simply control a multiplexer just before the internal transmit clock input. The DPLL output that may be used to feed the transmitter in FM modes lags by 90 the output of the DPLL used by the receiver. This makes the received and transmitted bit cells occur simultaneously, neglecting delays. A hardware reset selects the TRxC pin as the source of the transmit clock.

Table 4-6. Transmit Clock Source

Transmit Clock 1	Transmit Clock 0	
0	0	Transmit Clock = RTxC pin
0	1	Transmit Clock = TRxC pin
1	0	Transmit Clock = BR output
1	1	Transmit Clock = DPLL output

Bit 2: TRxC O/I

This bit determines the direction of the TRxC pin. If this bit is set to 1, the TRxC pin is an output and

carries the signal selected by D1 and D0 of this register. However, if either the receive or the transmit clock is programmed to come from the TRxC pin, TRxC will be an input, regardless of the state of this bit. The TRxC pin is also an input if this bit is set to 0. A hardware reset forces this bit to 0.

Bits 1 and 0: TRxC Output Source 1 and 0

These bits determine the signal to be echoed out of the SCC via the TRxC pin. No signal is produced if TRxC has been programmed as the source of either the receive or the transmit clock. If TRxC O/I (bit 2) is reset to 0, these bits are ignored.

If the XTAL oscillator output is programmed to be echoed, the XTAL oscillator has not been enabled, the TRxC pin goes High. The DPLL signal that is echoed is the DPLL signal used by the receiver. Hardware reset selects the XTAL oscillator as the output source.

Table 4-7. Transmit External Control Selection

Output Signal	Output Signal	
0	0	TRxC = XTAL oscillator output
0	1	TRxC = Transmit Clock
1	0	TRxC = BR output
1	1	TRxC = DPLL output (receive)

4.1.13 Write Register 12 (Lower Byte of Baud Rate Generator Time Constant)

WR12 contains the lower byte of the time constant for the baud rate generator. The time constant can be changed at any time, but the new value does not take effect until the next time the time constant is loaded into the down counter. No attempt is made to synchronize the loading of the time constant into WR12 and WR13 with the clock driving the down counter. For this reason, it is advisable to disable the baud rate generator while the new time constant is loaded into WR12 and WR13. Ordinarily, this is done anyway to prevent a load of the down counter between the writing of the upper and lower bytes of the time constant.

The formula for determining the appropriate time constant for a given baud rate is shown below with the desired rate in bits per seconds and the BR clock period in seconds. This formula is derived because the counter decrements from N down to 0-plus-one-cycle for reloading the time constant and is then fed to a toggle flip-flop to make the output a square wave. Bit positions for WR12 are shown in Figure 4-13.

Time constant =
[½ * desired rate *BR clock period)] −2

4.1.14 Write Register 13 (Upper Byte of Baud Rate Generator Time Constant)

WR13 contains the upper byte of the time constant for the baud rate generator. Bit positions for WR13 are shown in Figure 4-14.

4.1.15 Write Register 14 (Miscellaneous Control Bits)

WR14 contains some miscellaneous control bits. Bit positions for WR14 are shown in Figure 4-15.

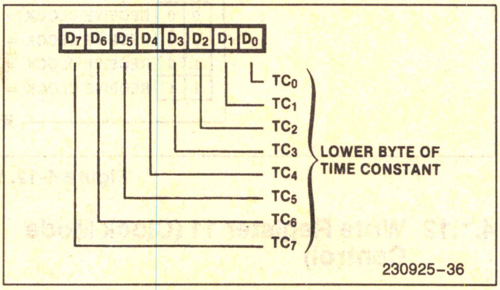


Figure 4-13. Write Register 12

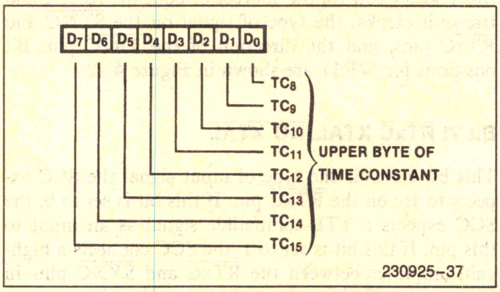


Figure 4-14. Write Register 13

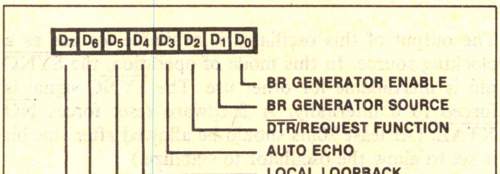


Figure 4-15. Write Register 14

Bits 7 and 5: Digital Phase-Locked Loop Command Bits

These three bits encode the eight commands for the Digital Phase-Locked Loop. A channel for hardware reset disables the DPLL, resets the missing clock latches, sets the source to the $\overline{RTxC}$ pin and selects NRZI mode. The Enter Search Mode command enables the DPLL after a reset.

Null Command (000). This command has no effect on the DPLL.

Enter Search Mode (001). Issuing this command causes the DPLL to enter the Search mode, where the DPLL searches for a locking edge in the incoming data stream. The action taken by the DPLL upon receipt of this command depends on the operating mode of the DPLL.

In NRZI mode, the output of the DPLL is High while the DPLL is waiting for an edge in the incoming data stream. After the Search mode is entered, the first edge the DPLL sees is assumed to be a valid data edge, and the DPLL begins the clock recovery operation from that point. The DPLL clock rate must be 32 times the data rate in NRZI mode. Upon leaving the Search mode, the first sampling edge of the DPLL occurs 16 of these 32X clocks after the first data edge and the second sampling edge occurs 48 of these 32X clocks after the first data edge. Beyond this point, the DPLL begins normal operation, adjusting the output to remain in sync with the incoming data.

In FM mode, the output of the DPLL is Low while the DPLL is waiting for an edge in the incoming data stream. The first edge the DPLL detects is assumed to be a valid clock edge. For this to be the case, the line must contain only clock edges; i.e. with FM1 encoding, the line must be continuous 0s. With FM0 encoding the line must be continuous 1s, whereas Manchester encoding requires alternating 1s and 0s on the line. The DPLL clock rate must be 16 times the data rate in FM mode. The DPLL output causes the receiver to sample the data stream in the nominal center of the two halves of the bit cell to decide whether the data was a 1 or a 0. After this command is issued, as in NRZI mode, the DPLL starts sampling immediately after the first edge is detected. (In FM mode, the DPLL examines the clock edge of every other bit cell to decide what correction must be made to remain in sync). If the DPLL does not see an edge during the expected window, the one clock missing bit in RR10 is set. If the DPLL does not see an edge after two successive attempts, the two clocks missing bit in RR10 is set and the DPLL automatically enters the Search mode. This command resets both clock missing latches.

Reset Clock Missing (010). Issuing this command resets the clock missing latches in RR10 and arms the latches to detect the next missing clock edge on the line. The clock missing latches are only used in FM mode.

Disable DPLL (011). Issuing this command disables the DPLL, resets the clock missing latches in RR10, and forces a continuous Search mode state.

Set Source = BR Gen (100). Issuing this command forces the clock for the DPLL to come from the output of the baud rate generator.

Set Source = $\overline{RTxC}$ (101). Issuing this command forces the clock for the DPLL to come from the $\overline{RTxC}$ pin or the crystal oscillator, depending on the state of the XTAL/no XTAL bit in WR11. This mode is selected by a channel or hardware reset.

Set FM Mode (110). This command forces the DPLL to operate in the FM mode and is used to recover the clock from FM or Manchester-encoded data (Manchester is decoded by placing the receiver in NRZ mode while the DPLL is in FM mode).

Set NRZI Mode (111). Issuing this command forces the DPLL to operate in the NRZI mode. This mode is also selected by a hardware or channel reset.

Bit 4: Local Loopback

Setting this bit to 1 selects the Local Loopback mode of operation. In this mode, the internal transmitted data is routed back to the receiver, as well as to the TxD pin. The $\overline{CTS}$ and $\overline{CD}$ inputs are ignored as enables in Local Loopback mode, even if auto enables is selected. (If so programmed, transitions on these inputs still cause interrupts.) This mode works with any Transmit/Receive mode except Loop mode. For meaningful results, the frequency of the transmit and receive clocks must be the same. This bit is reset by a channel or hardware reset.

Bit 3: Auto Echo

Setting this bit to 1 selects the Auto Echo mode of operation. In this mode, the TxD pin is connected to RxD, as in Local Loopback mode, but the receiver still listens to the RxD input. Transmitted data is never seen inside or outside the SCC in this mode, and $\overline{CTS}$ is ignored as a transmit enable. This bit is reset by a channel or hardware reset.

Bit 2: $\overline{\mathrm{DTR}}$/Request Function

This bit selects the function of the $\overline{\mathrm{DTR}}/\overline{\mathrm{REQ}}$ pin. If this bit is set to 0, the $\overline{\mathrm{DTR}}/\overline{\mathrm{REQ}}$ pin follows the state of the DTR bit in WR5. If this bit is set to 1, the $\overline{\mathrm{DTR}}/\overline{\mathrm{REQ}}$ pin goes Low whenever the transmit buffer becomes empty and in any of the synchronous mode when CRC has been sent at the end of a message. The request function on the $\overline{\mathrm{DTR}}/\overline{\mathrm{REQ}}$ pin differs somewhat from the transmit request function available on the $\overline{\mathrm{RDY}}/\overline{\mathrm{REQ}}$ pin in that $\overline{\mathrm{REQUEST}}$ does not go inactive until the internal operation satisfying the request is complete, which occurs four to five CLK cycles after the rising edge of $\overline{\mathrm{READ}}$ or $\overline{\mathrm{WRITE}}$. If the DMA used is edge-triggered, this difference is unimportant. This bit is reset by a channel or hardware reset.

Bit 1: Baud Rate Generator Source

This bit selects the source of the clock for the baud rate generator. If this bit is set to 0, the baud rate generator clock comes from either the $\overline{\mathrm{RTxC}}$ pin or the XTAL oscillator (depending on the state of the XTAL/no XTAL bit). If this bit is set to 1, the clock for the baud rate generator is the SCC's CLK input. Hardware reset sets this bit to 0, selecting the $\overline{\mathrm{RTxC}}$ pin as the clock source for the baud rate generator.

Bit 0: Baud Rate Generator Enable

This bit controls the operation of the baud rate generator. The counter in the baud rate generator is enabled for counting when this bit is set to 1, and counting is inhibited when this bit is set to 0. When this bit is set to 1, change in the state of this bit is not reflected by the output of the baud rate generator for two counts of the counter. This allows the command to be synchronized. However, when set to 0, disabling is immediate. This bit is reset by a hardware reset.

4.1.16 Write Register 15 (External/Status Interrupt Control)

WR15 is the External/Status Source Control register. If the External/Status interrupts are enabled as a group via WR1, bits in this register control which External/Status conditions can cause an interrupt. Only the External/Status conditions that occur after the controlling bit is set to 1 will cause an interrupt. This is true even if an External/Status condition is pending at the time the bit is set. Bit positions for WR15 are shown in Figure 4-16.

Bit 7: Break/Abort IE

If this bit is set to 1, a change in the Break/Abort status of the receiver causes an External/Status interrupt. This bit is set by a channel or hardware reset.

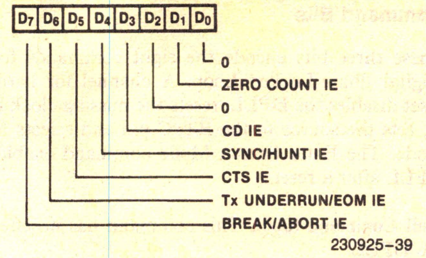


Figure 4-16. Write Register 15

Bit 6: Tx Underrun/EOM

If this bit is set to 1, a change of state by the Tx Underrun/EOM latch in the receiver causes an External/Status interrupt. This bit is set to 1 by a channel or hardware reset.

Bit 5: CTS IE

If this bit is set to 1, a change of state on the $\overline{\mathrm{CTS}}$ pin causes an External/Status interrupt. This bit is set by a channel or hardware reset.

Bit 4: Sync/Hunt IE

If this bit is set to 1, a change of state on the $\overline{\mathrm{SYNC}}$ pin causes an External/Status interrupt in Asynchronous mode, and a change of state in the Hunt bit in the receiver causes an External/Status interrupt in synchronous modes. This bit is set by a channel or hardware reset.

Bit 3: CD IE

If this bit is set to 1, a change of state on the $\overline{\mathrm{CD}}$ pin causes an External/Status interrupt. This bit is set by a channel or hardware reset.

Bit 2: Not Used. Must be 0.

Bit 1: Zero Count IE

If this bit is set to 1, an External/Status interrupt is generated whenever the counter in the baud rate generator reaches 0. This bit is set to 0 by a channel or hardware reset.

Bit 0: Not Used. Must be 0.

4.2 READ REGISTERS

The SCC contains nine read registers in each channel. The status of these registers is continually changing and depends on the mode of communication, received and transmitted data, and the manner in which this data is transferred to and from the CPU. The following description details the bit assignments for each register.

4.2.1 Read Register 0 (Transmit/Receive Buffer Status and External Status)

Read Register 0 contains the status of the receive and transmit buffers. RR0 also contains the status bits for the six sources of External/Status interrupts. The bit configuration is illustrated in Figure 4-17.

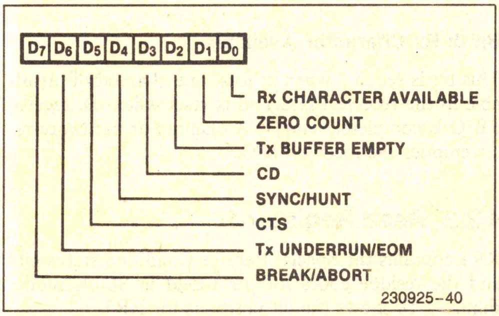


Figure 4-17. Read Register 0

Bit 7: Break/Abort

In the Asynchronous mode, this bit is set when a Break sequence (null character plus framing error) is detected in the receive data stream. This bit is reset when the sequence is terminated, leaving a single null character in the receive FIFO. This character should be read and discarded. In SDLC mode, this bit is set by the detection of an Abort sequence (seven or more 1s), then reset automatically at the termination of the Abort sequence. In either case, if the Break/Abort IE bit is set, an External/Status interrupt is initiated. Unlike the remainder of the External/status bits, both transitions are guaranteed to cause an External/Status interrupt, even if another External/Status interrupt is pending at the time these transitions occur. This procedure is necessary because Abort or Break conditions may not persist.

Bit 6: Tx Underrun/EOM

This bit is set by a channel or hardware reset and when the transmitter is disabled or a Send Abort command is issued. This bit can only be reset by the Reset Tx Underrun/EOM Latch command in WR0. When the Transmit Underrun occurs, this bit is set and causes an External/Status interrupt (if the Tx Underrun/EOM IE bit is set).

Only the 0-to-1 transition of this bit causes an interrupt. This bit is always 1 in Asynchronous mode, unless a reset Tx Underrun/EOM Latch command has been erroneously issued. In this case, the Send Abort command can be used to set the bit to one and at the same time cause an External/Status interrupt.

Bit 5: Clear to Send

If the CTS IE bit in WR15 is set, this bit indicates the state of the $\overline{\text{CTS}}$ pin the last time any of the enabled External/status bits changed. Any transition on the $\overline{\text{CTS}}$ pin while no interrupt is pending latches the state of the $\overline{\text{CTS}}$ pin and generates an External/Status interrupt. Any odd number of transitions on the $\overline{\text{CTS}}$ pin while another External/Status interrupt is pending also causes an External/Status interrupts condition. If the CTS IE bit is reset, it merely reports the current unlatched state of the $\overline{\text{CTS}}$ pin.

Bit 4: SYNC/HUNT

The operation of this bit is similar to that of the $\overline{\text{CTS}}$ bit, except that the condition monitored by the bit varies depending on the mode in which the SCC is operating.

When the XTAL oscillator option is selected in asynchronous modes, this bit is forced to 0 (no External/Status interrupt is generated). Selecting the XTAL oscillator in synchronous or SDLC modes had no effect on the operation of this bit. The XTAL oscillator should not be selected in External Sync mode.

In Asynchronous mode, the operation of this bit is identical to that of the CTS status bit, except that this bit reports the state of the $\overline{\text{SYNC}}$ pin.

In External sync mode the $\overline{\text{SYNC}}$ pin is used by external logic to signal character synchronization. When the Enter Hunt Mode command is issued in External Sync mode, the $\overline{\text{SYNC}}$ pin must be held High by the external sync logic until character synchronization is achieved. A High on the $\overline{\text{SYNC}}$ pin holds the Sync/Hunt bit in the reset condition.

When external synchronization is achieved, $\overline{\text{SYNC}}$ must be driven Low on the second rising edge of the Receive Clock after the last rising edge of the Receive Clock on which the last bit of the receive character was received. Once $\overline{\text{SYNC}}$ is forced Low, it is good

practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or that a new message is about to start. Both transitions on the $\overline{\text{SYNC}}$ pin cause External/Status interrupts if the Sync/Hunt IE bit is set to 1.

The Enter Hunt Mode command should be issued whenever character synchronization is lost. At the same time, the CPU should inform the external logic that character synchronization has been lost and that the SCC is waiting for $\overline{\text{SYNC}}$ to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode command. The Sync/Hunt bit is reset when the SCC established character synchronization. Both transitions cause External/Status interrupts if the Sync/Hunt IE bit is set. When the CPU detects the end of message or the loss of character synchronization, the Enter Hunt Mode command should be issued to set the Sync/Hunt bit and cause an External/Status interrupt. In this mode, the $\overline{\text{SYNC}}$ pin is an output, which goes Low every time a sync pattern is detected in the data stream.

In the SDLC modes, the Sync/Hunt bit is initially set by the Enter Hunt Mode command or when the receiver is disabled. It is reset when the opening flag of the first frame is detected by the SCC. An External/Status interrupt is also generated if the Sync/Hunt IE bit is set. Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in SDLC mode, it does not need to be set when the end of the frame is detected. The SCC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode command or by disabling the receiver.

Bit 3: Carrier Detect

If the CD IE bit in WR15 is set, this bit indicates the state of the $\overline{\text{CD}}$ pin the last time the Enabled External/status bits changed. Any transition on the $\overline{\text{CD}}$ pin while no interrupt is pending latches the state of the $\overline{\text{CD}}$ pin and generates an External/Status interrupt. Any odd number of transitions on the $\overline{\text{CD}}$ pin while another External/Status interrupt is pending also causes an External/Status interrupt condition. If the CD IE bit is reset, this bit merely reports the current, unlatched state of the $\overline{\text{CD}}$ pin.

Bit 2: Tx Buffer Empty

This bit is set to 1 when the transmit buffer is empty. It is reset while CRC is sent in a synchronous SDLC mode and while the transmit buffer is full. The bit is reset when a character is loaded into the transmit buffer. This bit is always in the set condition after a hardware or channel reset.

Bit 1: Zero Count

If the Zero Count Interrupt Enable bit is set in WR15, this bit is set to one while the counter in the baud rate generator is at the count of zero. If there is no other External/Status interrupt condition pending at the time this bit set, an External/Status interrupt is generated. However, if there is another External/Status interrupt pending at this time, no interrupt is initiated until interrupt service is complete. If the Zero Count condition does not persist beyond the end of the interrupt service routine, no interrupt will be generated. This bit is not latched High, even though the other External/Status latches close as a result of the Low-to-High transition on ZC. The interrupt service routine should check the other External/Status conditions for changes. If none changed, ZC was the source. In polled applications, check the IP bit in RR3A for a status change and then proceed as in the interrupt service routine.

Bit 0: Rx Character Available

This bit is set to 1 when at least one character is available in the receive FIFO and is reset when the receive FIFO is completely empty. A channel or hardware reset empties the receive FIFO.

4.2.2 Read Register 1

RR1 contains the Special Receive Condition status bits and the residue codes for the I-field in SDLC mode. Figure 4-18 shows the bit positions for RR1.

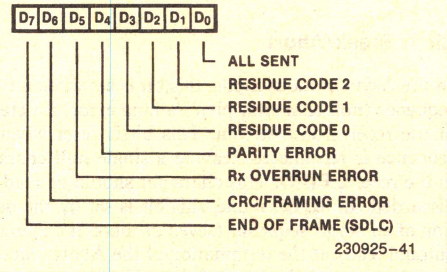


Figure 4-18. Read Register 1

Bit 7: End of Frame (SDLC)

This bit is used only in SDLC mode and indicates that a valid closing flag has been received and that the CRC Error bit and residue codes are valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame. This bit is reset in any mode other than SDLC.

Bit 6: CRC/Framing Error

If a framing error occurs (in Asynchronous mode), this bit is set (and not latched) for the receive character in

which the framing error occurred. Detection of a framing error adds an additional one-half bit to the character time so that the framing error is not interpreted as a new Start bit. In Synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command, but the bit is never latched. Therefore, it is always updated when the next character is received. When used for CRC error status in Synchronous or SDLC modes, this bit is usually set since most bit combinations, except for a correctly completed message, result in a non-zero CRC.

Bit 5: Receive Overrun Error

This bit indicates that the receive FIFO has overflowed. Only the character that has been written over is flagged with this error, and when the character is read, the Error condition is latched until reset by the Error Reset command. The overrun character and all subsequent characters received until the Error Reset command is issued causes a Special Receive Condition vector to be returned.

Bit 4: Parity Error

When parity is enabled, this bit is set for the characters whose parity does not match the programmed sense (even/odd). This bit is latched so that once an error occurs, it remains set until the Error Reset command is issued. If the parity in Special Condition bit is set, a parity error causes a Special Receive Condition vector to be returned on the character containing the error and on all subsequent characters until the Error Reset command is issued.

Bits 3–1: Residue Codes 2, 1, and 0

In those cases in SDLC mode where the received I-Field is not an integral multiple of the character length, these three bits indicate the length of the I-Field and are meaningful only for the transfer in which the end of frame bit is set. This field is set to 011 by a channel or hardware reset and is forced to this state in Asynchronous mode. These three bits can leave this state only if SDLC is selected and a character is received. The codes signify the following (Reference Table 4-8) when a receive character length is eight bits per character.

I-Field bits are right-justified in all cases. If a receive character length other than eight bits is used for the I-Field, a table similar to Table 4-8 can be constructed for each different character length. Table 4-9 shows the residue codes for no residue (the I-Field boundary lies on a character boundary).

Table 4-8. I-Field Bit Selection (8 Bits Only)

Residue	Residue	Residue	I-Field Bits in Previous Byte	I-Field Bits in Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

Table 4-9. Residue Bits/Character

Bits/Char	Residue	Residue	Residue
8	0	1	1
7	0	0	0
6	0	1	0
5	0	0	1

Bit 0: All Sent

In Asynchronous mode, this bit is set when all characters have completely cleared the transmitter. Most modems contains additional delay in the data path, which requires the modem control signals to remain active until after the data has cleared both the transmitter and the modem. This bit is always set in synchronous and SDLC modes.

4.2.3 Read Register 2

RR2 contains the interrupt vector written into WR2. In the Non-Vectored Mode (WR9 bit D5 = 1), reading this register (channel A or B) causes the IUS bit of the highest priority IP (Interrupt Pending) to be set. In the Vectored Mode (WR9 bit D5 = 0), reading this register does not effect the internal IUS logic. When the register is accessed in Channel A, the vector returned is the vector actually stored in WR2. When this register is accessed in Channel B, the vector returned includes status information in bits 1, 2, and 3 or in bits 6, 5, and 4, depending on the state of the Status High/Status Low bit in WR9 and independent of the state of the VIS bit in WR9. The vector is modified according to Table 4-3 shown in the explanation of the VIS bit in WR9. If no interrupts are pending, the status is V3, V2, V1 = 011, or V6, V5, V4 = 110. Figure 4-19 shows the bit positions for RR2.

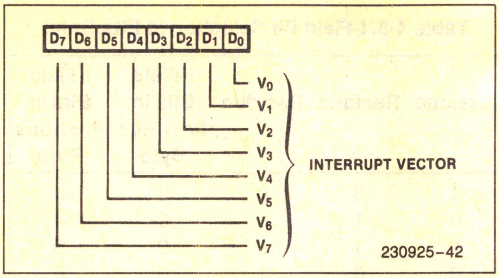


Figure 4-19. Read Register 2

4.2.4 Read Register 3

RR3 is the Interrupt Pending register. The status of each of the Interrupt Pending bits in the SCC is reported in this register. This register exists only in Channel A. If this register is accessed in Channel B, all 0s are returned. The two unused bits are always returned as 0. Figure 4-20 shows the bit position for RR3.

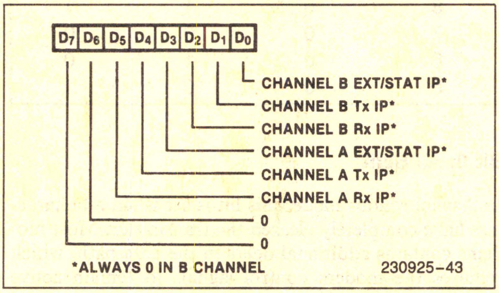


Figure 4-20. Read Register 3

4.2.5 Read Register 8

RR8 is the Receive Data register.

4.2.6 Read Register 10

RR10 contains some miscellaneous status bits. Unused bits are always 0. Bit positions for RR10 are shown in Figure 4-21.

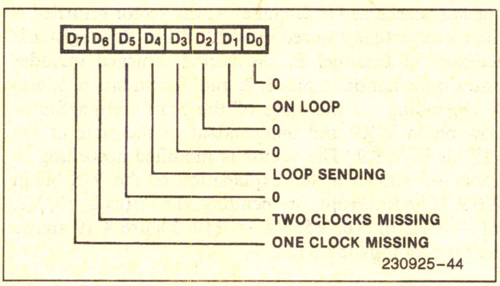


Figure 4-21. Read Register 10

Bit 7: One Clock Missing

While operating in the FM mode, the DPLL sets this bit to 1 when it does not see a clock edge on the incoming line in the window where it expects one. This bit is latched until reset by a Reset Missing Clock or Enter Search Mode command in WR10. In the NRZI mode of operation and while the DPLL is disabled, this bit is always 0.

Bit 6: Two Clocks Missing

While operating in the FM mode, the DPLL sets this bit to 1 when it does not see a clock edge in two successive tries. At the same time the DPLL enters the Search mode. This bit is latched until reset by a Reset Missing Clock or Enter Search Mode command in WR10. In the NRZI mode of operation and while the DPLL is disabled, this bit is always 0.

Bit 4: Loop Sending

This bit is set to 1 in SDLC Loop mode while the transmitter is in control of the loop, that is, while the SCC is actively transmitting on the loop. This bit is reset at all other times.

Bit 1: On Loop

This bit is set to 1 while the SCC is actually on-loop in SDLC Loop mode. This bit is set to 1 in the X.21 mode (Loop mode selected while in monosync) when the transmitter goes active. This bit is 0 at all other times.

4.2.7 Read Register 12

RR12 returns the value stored in WR12, the lower byte of the time constant for the baud rate generator. Figure 4-22 shows the bit positions for RR12.

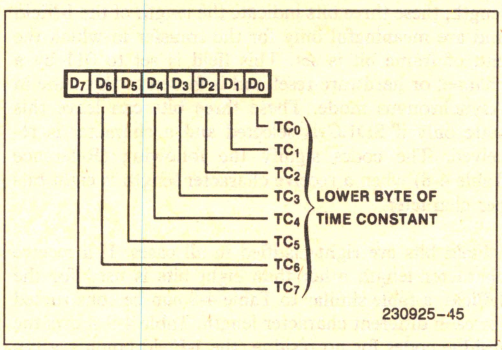


Figure 4-22. Read Register 12

4.2.8 Read Register 13

RR13 returns the value stored in WR13, the upper byte of the time constant for the baud rate generator. Figure 4-23 shows the bit positions for RR13.

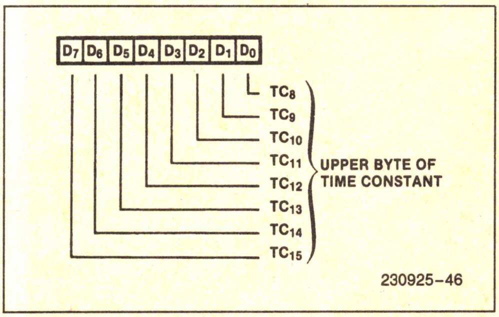


Figure 4-23. Read Register 13

4.2.9 Read Register 15

RR15 reflects the value stored in WR15, the External/Status IE bits. The two unused bits are always returned as 0s. Figure 4-24 shows the bits positions for RR15.

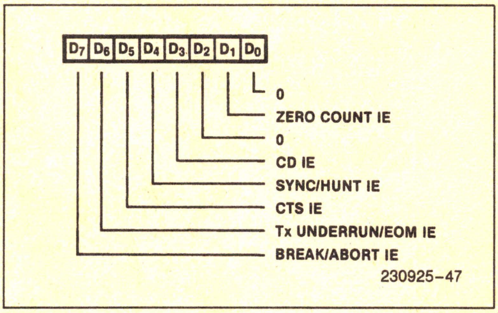


Figure 4-24. Read Register 15

Other Components 3

8291A
GPIB TALKER/LISTENER

- Designed to Interface Microprocessors (e.g., 8048/49, 8051, 8080/85, 8086/88) to an IEEE Standard 488 Digital Interface Bus
- Programmable Data Transfer Rate
- Complete Source and Acceptor Handshake
- Complete Talker and Listener Functions with Extended Addressing
- Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local Functions
- Selectable Interrupts
- On-Chip Primary and Secondary Address Recognition
- Automatic Handling of Addressing and Handshake Protocol
- Provision for Software Implementation of Additional Features
- 1–8 MHz Clock Range
- 16 Registers (8 Read, 8 Write), 2 for Data Transfer, the Rest for Interface Function Control, Status, etc.
- Directly Interfaces to External Non-Inverting Tranceivers for Connection to the GPIB
- Provides Three Addressing Modes, Allowing the Chip to be Addressed Either as a Major or a Minor Talker/Listener with Primary or Secondary Addressing
- DMA Handshake Provision Allows for Bus Transfers without CPU Intervention
- Trigger Output Pin
- On-Chip EOS (End of Sequence) Message Recognition Facilitates Handling of Multi-Byte Transfers

The 8291A is an enhanced version of the 8291 GPIB Talker/Listener designed to interface microprocessors to an IEEE Standard 488 Instrumentation Interface Bus. It implements all of the Standard's interface functions except for the controller. The controller function can be added with the 8292 GPIB Controller, and the 8293 GPIB Transceiver performs the electrical interface for Talker/Listener and Talker/Listener/Controller configurations.

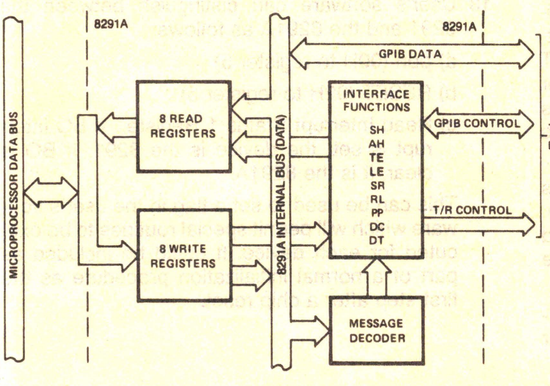


Figure 1. Block Diagram

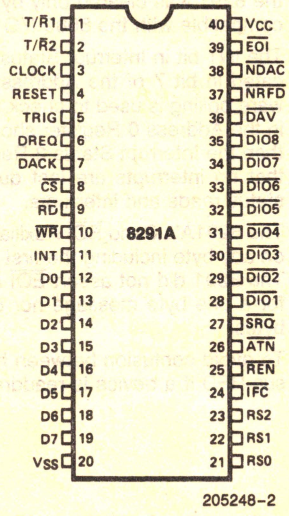


Figure 2. Pin Configuration

November 1986
Order Number: 205248-002

8291A FEATURES AND IMPROVEMENTS

The 8291A is an improved design of the 8291 GPIB Talker/Listener. Most of the functions are identical to the 8291, and the pin configuration is unchanged.

The 8291A offers the following improvements to the 8291:

1. $\overline{\text{EOI}}$ is active with the data as a ninth data bit rather than as a control bit. This is to comply with some additions to the 1975 IEEE-488 Standard incorporated in the 1978 Standard.
2. The BO interrupt is not asserted until RFD is true. If the Controller asserts $\overline{\text{ATN}}$ synchronously, the data is guaranteed to be transmitted. If the Controller asserts $\overline{\text{ATN}}$ asynchronously, the SH (Source Handshake) will return to SIDS (Source Idle State), and the output data will be cleared. Then, if $\overline{\text{ATN}}$ is released while the 8291A is addressed to talk, a new BO interrupt will be generated. This change fixes 8291 problems which caused data to be lost or repeated and a problem with the RQS bit (sometimes cannot be asserted while talking).
3. LLOC and REMC interrupts are setting flipflops rather than toggling flipflops in the interrupt backup register. This ensures that the CPU knows that these state changes have occurred. The actual state can be determined by checking the LLO and REM status bits in the upper nibble of the Interrupt Status 2 Register.
4. DREQ is cleared by $\overline{\text{DACK}}$ ($\overline{\text{RD}}$ + $\overline{\text{WR}}$). DREQ on the 8291 was cleared only by $\overline{\text{DACK}}$ which is not compatible with the 8089 I/O Processor.
5. The INT bit in Interrupt Status 2 Register is duplicated in bit 7 of the Address 0 Register. If software polling is used to check for an interrupt, INT in the Address 0 Register should be polled rather than the Interrupt Status 2 Register. This ensures that no interrupts are lost due to asynchronous status reads and interrupts.
6. The 8291A's Send $\overline{\text{EOI}}$ Auxiliary Command works on any byte including the first byte of a message. The 8291 did not assert $\overline{\text{EOI}}$ after this command for a one byte message nor on two consecutive bytes.
7. To avoid confusion between holdoff on DAV versus RFD if a device is readdressed from a talker to a listener role or vice-versa during a holdoff, the "Holdoff on Source Handshake" has been eliminated. Only "Holdoff on Acceptor Handshake" is available.
8. The rsv local message is cleared automatically upon exit from SPAS if (APRS:STRS:SPAS) occurred. The automatic resetting of the bit after the serial poll is complete simplifies the service request software.
9. The SPASC interrupt on the 8291 has been replaced by the SPC (Serial Poll Complete) interrupt on the 8291A. SPC interrupt is set on exit from SPAS if APRS:STRS:SPAS occurred, indicating that the controller has read the bus status byte after the 8291A requested service. The SPASC interrupt was ambiguous because a controller could enter SPAS and exit SPAS generating two SPASC interrupts without reading the serial poll status byte. The SPC interrupt also simplifies the CPU's software by eliminating the interrupt when the serial poll is half way done.
10. The rtl Auxiliary Command in the 8291 has been replaced by Set and Clear rtl Commands in the 8291A. Using the new commands, the CPU has the flexibility to extend the length of local mode or leave it as a short pulse as in the 8291.
11. A holdoff RFD on GET, SDC, and DCL feature has been added to prevent additional bus activity while the CPU is responding to any of these commands. The feature is enabled by a new bit (B_4) in the Auxiliary Register B.
12. On the 8291, BO could cease to occur upon $\overline{\text{IFC}}$ going false if $\overline{\text{IFC}}$ occurred asynchronously. On the 8291A, BO continues to occur after $\overline{\text{IFC}}$ has gone false even if it arrived asynchronously.
13. User's software can distinguish between the 8291 and the 8291A as follows:

    a) pon (00H to register 5)

    b) RESET (02H to register 5)

    c) Read Interrupt Status 1 Register. If BO interrupt is set, the device is the 8291. If BO is clear, it is the 8291A.

    This can be used to set a flag in the user's software which will permit special routines to be executed for each device. It could be included as part of a normal initialization procedure as the first step after a chip reset.

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_0–D_7	12–19	I/O	**DATA BUS PORT:** To be connected to microprocessor data bus.
RS_0–RS_2	21–23	I	**REGISTER SELECT:** Inputs, to be connected to three nonmultiplexed microprocessor address bus lines. Select which of the 8 internal read (write) registers will be read from (written into) with the execution of $\overline{RD}$ ($\overline{WR}$).
$\overline{CS}$	8	I	**CHIP SELECT:** When low, enables reading from or writing into the register selected by RS_0–RS_2.
$\overline{RD}$	9	I	**READ STROBE:** When low with $\overline{CS}$ or $\overline{DACK}$ low, selected register contents are read.
$\overline{WR}$	10	I	**WRITE STROBE:** When low with $\overline{CS}$ or $\overline{DACK}$ low, data is written into the selected register.
INT ($\overline{INT}$)	11	O	**INTERRUPT REQUEST:** To the microprocessor, set high for request and cleared when the appropriate register is accessed by the CPU. May be software configured to be active low.
DREQ	6	O	**DMA REQUEST:** Normally low, set high to indicate byte output or byte input in DMA mode; reset by $\overline{DACK}$.
$\overline{DACK}$	7	I	**DMA ACKNOWLEDGE:** When low, resets DREQ and selects data in/data out register for DMA data transfer (actual transfer done by $\overline{RD}$/$\overline{WR}$ pulse). Must be high if DMA is not used.
TRIG	5	O	**TRIGGER OUTPUT:** Normally low; generates a triggering pulse with 1 μsec min. width in response to the GET bus command or Trigger auxiliary command.
CLOCK	3	I	**EXTERNAL CLOCK:** Input, used only for T, delay generator. May be any speed in 1–8 MHz range.
RESET	4	I	**RESET INPUT:** When high, forces the device into an "idle" (initialization) mode. The device will remain at "idle" until released by the microprocessor, with the "Immediate Execute pon" local message.
$\overline{DIO_1}$–$\overline{DIO_8}$	28–35	I/O	**8-BIT GPIB DATA PORT:** Used for bidirectional data byte transfer between 8291A and GPIB via non-inverting external line transceivers.
$\overline{DAV}$	36	I/O	**DATA VALID:** GPIB handshake control line. Indicates the availability and validity of information on the $\overline{DIO_1}$–$\overline{DIO_8}$ and $\overline{EOI}$ lines.
$\overline{NRFD}$	37	I/O	**NOT READY FOR DATA:** GPIB handshake control line. Indicates the condition of readiness of device(s) connected to the bus to accept data.
$\overline{NDAC}$	38	I/O	**NOT DATA ACCEPTED:** GPIB handshake control line. Indicates the condition of acceptance of data by the device(s) connected to the bus.
$\overline{ATN}$	26	I	**ATTENTION:** GPIB command line. Specifies how data on $\overline{DIO}$ lines are to be interpreted.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{IFC}$	24	I	**INTERFACE CLEAR:** GPIB command line. Places the interface functions in a known quiescent state.
$\overline{SRQ}$	27	O	**SERVICE REQUEST:** GPIB command line. Indicates the need for attention and requests an interruption of the current sequence of events on the GPIB.
$\overline{REN}$	25	I	**REMOTE ENABLE:** GPIB command line. Selects (in conjunction with other messages) remote or local control of the device.
$\overline{EOI}$	39	I/O	**END OR IDENTITY:** GPIB command line. Indicates the end of a multiple byte transfer sequence or, in conjunction with $\overline{ATN}$, addresses the device during a polling sequence.
$T/\overline{R}1$	1	O	**EXTERNAL TRANSCEIVERS CONTROL LINE:** Set high to indicate output data/signals on the $\overline{DIO_1}-\overline{DIO_8}$ and $\overline{DAV}$ lines and input signals on the $\overline{NRFD}$ and $\overline{NDAC}$ lines (active source handshake). Set low to indicate input data/signals on the $\overline{DIO_1}-\overline{DIO_8}$ and $\overline{DAV}$ lines and output signals on the $\overline{NRFD}$ and $\overline{NDAC}$ lines (active acceptor handshake).
$T/\overline{R}2$	2	O	**EXTERNAL TRANSCEIVERS CONTROL LINE:** Set to indicate output signals on the $\overline{EOI}$ line. Set low to indicate expected input signal on the $\overline{EOI}$ line during parallel poll.
V_{CC}	40	P.S.	**POSITIVE POWER SUPPLY:** (5V ±10%).
GND	20	P.S.	**CIRCUIT GROUND POTENTIAL.**

NOTE:
All signals on the 8291A pins are specified with positive logic. However, IEEE 488 specifies negative logic on its 16 signal lines. Thus, the data is inverted once from D_0-D_7 to $\overline{DIO_0}-\overline{DIO_8}$ and non-inverting bus transceivers should be used.

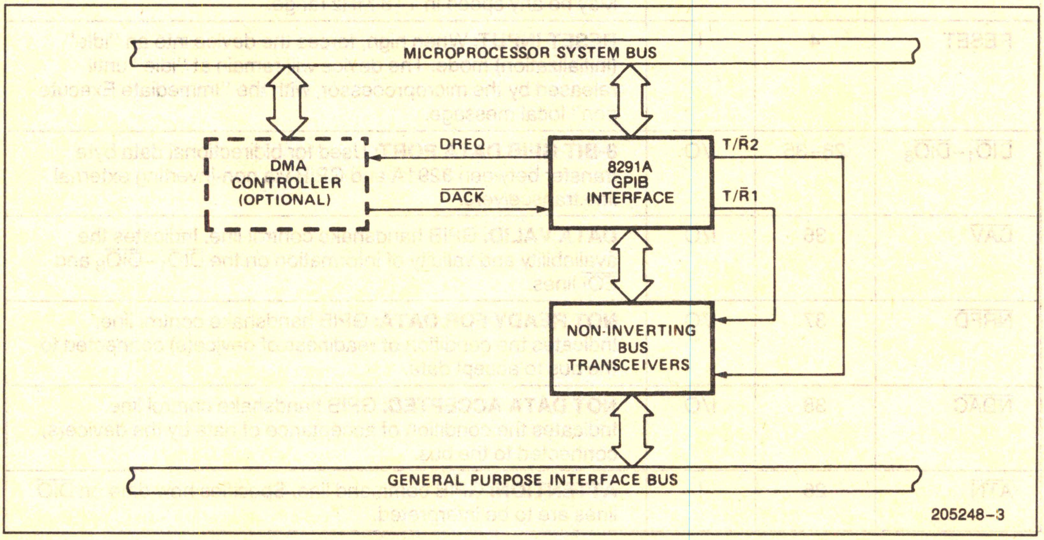


Figure 3. 8291A System Diagram

THE GENERAL PURPOSE INTERFACE BUS (GPIB)

The General Purpose Interface Bus (GPIB) is defined in the IEEE Standard 488-1978 "Digital Interface for Programmable Instrumentation." Although a knowledge of this standard is assumed, Figure 4 provides the bus structure for quick reference. Also, Tables 2 and 3 reference the interface state mnemonics and the interface messages respectively. Modified state diagrams for the 8291A are presented in Appendix A.

General Description

The 8291A is a microprocessor-controlled device designed to interface microprocessors, e.g., 8048/49, 8051, 8080/85, 8086/88 to the GPIB. It implements all of the interface functions defined in the IEEE-488 Standard except for the controller function. If an implementation of the Standard's Controller is desired, it can be connected with an Intel® 8292 to form a complete interface.

The 8291A handles communication between a microprocessor-controlled device and the GPIB. Its capabilities include data transfer, handshake protocol, talker/listener addressing procedures, device clearing and triggering, service request, and both serial and parallel polling. In most procedures, it does not disturb the microprocessor unless a byte has arrived (input buffer full) or has to be sent out (output buffer empty).

The 8291A architecture includes 16 registers. Eight of these registers may be written into by the microprocessor. The other eight registers may be read by the microprocessor. One each of these read and write registers is for direct data transfers. The rest of the write registers control the various features of the chip, while the rest of the read registers provide the microprocessor with a monitor of GPIB states, various bus conditions, and device conditions.

GPIB Addressing

Each device connected to the GPIB must have at least one address whereby the controller device in charge of the bus can configure it to talk, listen, or send status. An 8291A implementation of the GPIB offers the user three alternative addressing modes for which the device can be initialized for each application. The first of these modes allows for the device to have two separate primary addresses. The second mode allows the user to implement a single talker/listener with a two byte address (primary address + secondary address). The third mode again allows for two distinct addresses but in this instance, they can each have a ten-bit address (5 low-order bits of each of two bytes). However, this mode requires that the secondary addresses be passed to the microprocessor for verification. These three addressing schemes are described in more detail in the discussion of the Address Registers.

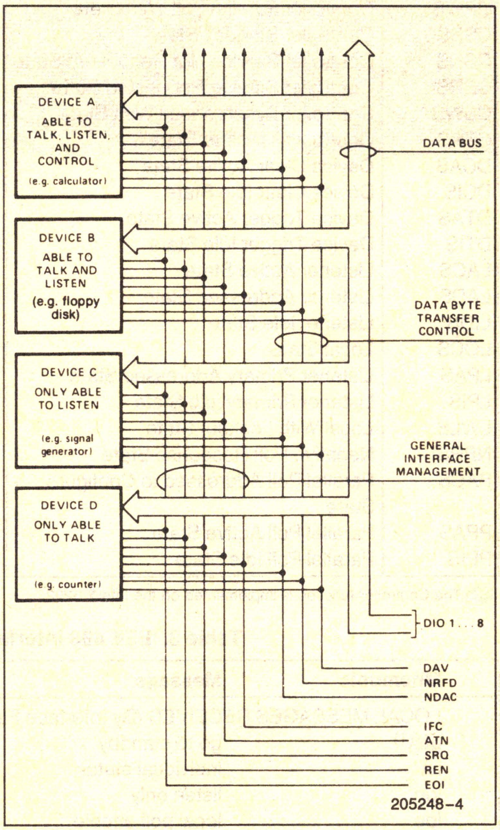


Figure 4. Interface Capabilities and Bus Structure

Table 2. IEEE 488 Interface State Mnemonics

Mnemonic	State Represented
ACDS	Accept Data State
ACRS	Acceptor Ready State
AIDS	Acceptor Idle State
ANRS	Acceptor Not Ready State
APRS	Affirmative Poll Response State
AWNS	Acceptor Wait for New Cycle State
CACS	Controller Active State
CADS	Controller Addressed State
CAWS	Controller Active Wait State
CIDS	Controller Idle State
CPPS	Controller Parallell Poll State
CPWS	Controller Parallel Poll Wait State
CSBS	Controller Standby State
CSNS	Controller Service Not Requested State
CSRS	Controller Service Requested State
CSWS	Controller Synchronous Wait State
CTRS	Controller Transfer State
DCAS	Device Clear Active State
DCIS	Device Clear Idle State
DTAS	Device Trigger Active State
DTIS	Device Trigger Idle State
LACS	Listener Active State
LADS	Listener Addressed State
LIDS	Listener Idle State
LOCS	Local State
LPAS	Listener Primary Addressed State
LPIS	Listener Primary Idle State
LWLS	Local With Lockout State
NPRS	Negative Poll Response State
PACS	Parallel Poll Addressed to Configure State
PPAS	Parallel Poll Active State
PPIS	Parallel Poll Idle State
PPSS	Parallel Poll Standby State
PUCS	Parallel Poll Unaddressed to Configure State
REMS	Remote State
RWLS	Remote With Lockout State
SACS	System Control Active State
SDYS	Source Delay State
SGNS	Source Generate State
SIAS	System Control Interface Clear Active State
SIDS	Source Idle State
SIIS	System Control Interface Clear Idle State
SINS	System Control Interface Clear Not Active State
SIWS	Source Idle Wait State
SNAS	System Control Not Active State
SPAS	Serial Poll Active State
SPIS	Serial Poll Idle State
SPMS	Serial Poll Mode State
SRAS	System Control Remote Enable Active State
SRIS	System Control Remote Enable Idle State
SRNS	System Control Remote Enable Not Active State
SRQS	Service Request State
STRS	Source Transfer State
SWNS	Source Wait for New Cycle State
TACS	Talker Active State
TADS	Talker Addressed State
TIDS	Talker Idle State
TPIS	Talker Primary Idle State

The Controller function is implemented on the Intel® 8292.

Table 3. IEEE 488 Interface Message Reference List

Mnemonic	Message	Interface Function(s)
LOCAL MESSAGES RECEIVED (By Interface Functions)		
gts(1)	go to standby	C
ist	individual status	PP
lon	listen only	L, LE
lpe	local poll enable	PP
nba	new byte available	SH
pon	power on	SH, AH, T, TE, L, LE, SR, RL, PP, C
rdy	ready	AH
rpp(1)	request parallel poll	C
rsc(1)	request system control	C
rsv	request service	SR
rtl	return to local	RL
sic(1)	send interface clear	C
sre(1)	send remote enable	C
tca(1)	take control asynchronously	C

Table 3. IEEE 488 Interface Message Reference List (Continued)

Mnemonic	Message	Interface Function(s)
tcs(1)	take control synchronously	AH, C
ton	talk only	T, TE
REMOTE MESSAGES RECEIVED		
ATN	Attention	SH, AH, T, TE, L, LE, PP, C
DAB	Data Byte	(Via L, LE)
DAC	Data Accepted	SH
DAV	Data Valid	AH
DCL	Device Clear	DC
END	End	(via L, LE)
GET	Group Execute Trigger	DT
GTL	Go to Local	RL
IDY	Identify	L, LE, PP
IFC	Interface Clear	T, TE, L, LE, C
LLO	Local Lockout	RL
MLA	My Listen Address	L, LE, RL, T, TE
MSA	My Secondary Address	TE, LE, RL
MTA	My Talk Address	T, TE, L, LE
OSA	Other Secondary Address	TE
OTA	Other Talk Address	T, TE
PCG	Primary Command Group	TE, LE, PP
PPC(2)	Parallel Poll Configure	PP
[PPD](2)	Parallel Poll Disable	PP
[PPE](2)	Parallel Poll Enable	PP
PPR_N(1)	Parallel Poll Response N	(via C)
PPU(2)	Parallel Poll Unconfigure	PP
REN	Remote Enable	RL
RFD	Ready for Data	SH
RQS	Request Service	(via L, LE)
[SDC]	Select Device Clear	DC
SPD	Serial Poll Disable	T, TE
SPE	Serial Poll Enable	T, TE
SQR(1)	Service Request	(via C)
STB	Status Byte	(via L, LE)
TCT or [TCT](1)	Take Control	C
UNL	Unlisten	L, LE
REMOTE MESSAGES SENT		
ATN	Attentions	C
DAB	Data Byte	(Via T,TE)
DAC	Data Accepted	AH
DAV	Data Valid	SH
DCL	Device Clear	(via C)
END	End	(via T)
GET	Group Execute Trigger	(via C)
GTL	Go to Local	(via C)
IDY	Identify	C
IFC	Interface Clear	C
LLO	Local Lockout	(via C)
MLA or [MLA]	My Listen Address	(via C)
MSA or [MSA]	My Secondary Address	(via C)
MTA or [MTA]	My Talk Address	(via C)
OSA	Other Secondary Address	(via C)

Table 3. IEEE 488 Interface Message Reference List (Continued)

Mnemonic	Message	Interface Function(s)[3]
OTA	Other Talk Address	(via C)
PCG	Primary Command Group	(via C)
PPC	Parallel Poll Configure	(via C)
[PPD]	Parallel Poll Disable	(via C)
[PPE]	Parallel Poll Enable	(via C)
PPR_N	Parallel Poll Response N	PP
PPU	Parallel Poll Unconfigure	(via C)
REN	Remote Enable	C
RFD	Ready for Data	AH
RQS	Request Service	T, TE
[SDC]	Selected Device Clear	(via C)
SPD	Serial Poll Disable	(via C)
SPE	Serial Poll Enable	(via C)
SRQ	Service Request	SR
STB	Status Byte	(via T,TE)
TCT	Take Control	(via C)
UNL	Unlisten	(via C)

NOTES:
1. These messages are handled only by Intel's 8292.
2. Undefined commands which may be passed to the microprocessor.
3. All Controller messages must be sent via Intel's 8292.

8291A Registers

A bit-by-bit map of the 16 registers on the 8291A is presented in Figure 5. A more detailed explanation of each of these registers and their functions follows. The access of these registers by the microprocessor is accomplished by using the $\overline{CS}$, $\overline{RD}$, $\overline{WR}$, and RS_0–RS_2 pins.

Register	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	RS_0–RS_2
All Read Registers	0	0	1	CCC
All Write Registers	0	1	0	CCC
High Impedance	1	d	d	ddd

Data Registers

D17	D16	D15	D14	D13	D12	D11	D10

DATA-IN REGISTER (0R)

DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

DATA-OUT REGISTER (0W)

The Data-In Register is used to move data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. Incoming information is separately latched by this register, and its contents are not destroyed by a write to the data-out register. The RFD (Ready for Data) message is held false until the byte is removed from the data in register, either by the microprocessor or by DMA. The 8291A then completes the handshake automatically. In RFD holdoff mode (see Auxiliary Register A), the handshake is not finished until a command is sent telling the 8291A to release the holdoff. In this way, the same byte may be read several times, or an over anxious talker may be held off until all available data has been processed.

When the 8291A is addressed to talk, it uses the data-out register to move data onto the GPIB. After the BO interrupt is received and a byte is written to this register, the 8291A initiates and completes the handshake while sending the byte out over the bus. In the BO interrupt disable mode, the user should wait until BO is active before writing to the register. (In the DMA mode, this will happen automatically.) A read of the Data-In Register does not destroy the information in the Data-Out Register.

Interrupt Registers

CPT	APT	GET	END	DEC	ERR	BO	BI

INTERRUPT STATUS 1 (1R)

INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC

INTERRUPT STATUS 2 (2R)

CPT	APT	GET	END	DEC	ERR	BO	BI

INTERRUPT ENABLE 1 (1W)

0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC

INTERRUPT ENABLE 2 (2W)

INT	DTO	DLO	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0

ADDRESS 0 REGISTER

Figure 5. 8291A Registers

READ REGISTERS								REGISTER SELECT CODE			WRITE REGISTERS							
								RS2	RS1	RS0								
D17	D16	D15	D14	D13	D12	D11	D10	0	0	0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
DATA IN											DATA OUT							
CPT	APT	GET	END	DEC	ERR	BO	BI	0	0	1	CPT	APT	GET	END	DEC	ERR	BO	BI
INTERRUPT STATUS 1											INTERRUPT ENABLE 1							
INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC	0	1	0	0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC
INTERRUPT STATUS 2											INTERRUPT ENABLE 2							
S8	SEQS	S6	S5	S4	S3	S2	S1	0	1	1	S8	rsv	S6	S5	S4	S3	S2	S1
SERIAL POLL STATUS											SERIAL POLL MODE							
ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN	1	0	0	TO	LO	0	0	0	0	ADM1	ADM0
ADDRESS STATUS											ADDRESS MODE							
CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	1	0	1	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
COMMAND PASS THROUGH											AUX MODE							
INT	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0	1	1	0	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1
ADDRESS 0											ADDRESS 0/1							
X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1	1	1	1	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
ADDRESS 1											EOS							

The 8291A can be configured to generate an interrupt to the microprocessor upon the occurrence of any of 12 conditions or events on the GPIB. Upon receipt of an interrupt, the microprocessor must read the Interrupt Status Registers to determine which event has occurred, and then execute the appropriate service routine (if necessary). Each of the 12 interrupt status bits has a matching enable bit in the interrupt enable registers. These enable bits are used to select the events that will cause the INT pin to be asserted. Writing a logic "1" into any of these bits enables the corresponding interrupt status bits to generate an interrupt. Bits in the Interrupt Status Registers are set regardless of the states of the enable bits. The Interrupt Status Registers are then cleared upon being read or when a local pon (power-on) message is executed. If an event occurs while one of the Interrupt Status Registers is being read, the event is held until after its register is cleared and then placed in the register.

NOTE:
Reading the interrupt status registers clears the bits which were set. The software must examine *all* relevant bits in the interrupt status registers before disregarding the value or an important interrupt may be missed.

The mnemonics for each of the bits in these registers and a brief description of their respective functions appears in Table 4. This table also indicates how each of the interrupt bits is set.

NOTE:
The INT bit in the Address 0 Register is a duplicate of the INT bit in the Interrupt Status 2 Register. It is only a status bit. It does not generate interrupts and thus does not have a corresponding enable bit.

The BO and BI interrupts enable the user to perform data transfer cycles. BO indicates that a data byte should be written to the Data Out Register. It is set by TACS • (SWNS + SGNS) • RFD. It is reset when the data byte is written, ATN is asserted, or the 8291A exits TACS. Data should never be written to the Data Out Register before BO is set. Similarly, BI is set when an input byte is accepted into the 8291A and reset when the microprocessor reads the Data In Register. BO and BI are also reset by pon (power-on local message) and by a read of the Interrupt Status 1 Register. However, if it is so desired, data transfer cycles may be performed without reading the Interrupt Status 1 Register if all interrupts except for BO or BI are disabled; BO and BI will automatically reset after each byte is transferred.

Table 4. Interrupt Bits

	Bit	
Indicates Undefined Commands	CPT	An undefined command has been received.
Set by (TPAS + LPAS)•SCG•ACDS•MODE 3	APT	A secondary address must be passed through to the microprocessor for recognition.
Set by DTAS	GET	A group execute trigger has occurred.
Set by (EOS + EOI)•LACS	END	An EOS or EOI message has been received.
Set by DCAS	DEC	Device Clear Active State has occurred.
Set by TACS•nba•DAC•RFD	ERR	Interface error has occurred; no listeners are active.
TACS•(SWNS + SGNS)	BO	A byte should be output.
Set by LACS•ACDS	BI	A byte has been input.
Shows status of the INT pin	INT	These are status only. They will <u>not</u> generate interrupts, nor do they have corresponding mask bits.
The device has been enabled for a serial poll	SPAS	
The device is in local lock out state. (LWLS+RWLS)	LLO	
The device is in a remote state. (REMS+RWLS)	REM	
SPAS →$\overline{SPAS}$ if APRS:STRS:SPAS was true	SPC	Serial Poll Complete interrupt.
LLO ⇄ NO LLO	LLOC	Local lock out change interrupt.
Remote ⇄ Local	REMC	Remote/Local change interrupt.
Addressed ⇄ Unaddressed	ADSC	Address status change interrupt.[1]

205248–24

NOTE:
1. In ton (talk-only) and lon (listen-only) modes, no ADSC interrupt is generated.

If the 8291A is used in the interrupt mode, the INT and DREQ pins can be dedicated to data input and output interrupts respectively by enabling BI and DMAO, provided that no other interrupts are enabled. This eliminates the need to read the interrupt status registers if a byte is received or transmitted.

The ERR bit is set to indicate the bus error condition when the 8291A is an active talker and tries to send a byte to the GPIB, but there are no active listeners (e.g., all devices on the GPIB are in AIDS). The logical equivalent of (nba • TACS • DAC • RFD) will set this bit.

The DEC bit is set whenever DCAS has occurred. The user must define a known state to which all device functions will return in DCAS. Typically this state will be a power-on state. However, the state of the device functions at DCAS is at the designer's discretion. It should be noted that DCAS has no effect on the interface functions which are returned to a known state by the IFC (interface clear) message or the pon local message.

The END interrupt bit may be used by the microprocessor to detect that a multi-byte transfer has been completed. The bit will be set when the 8291A is an active listener (LACS) and either EOS (provided the End on EOS Received feature is enabled in the Auxiliary Register A) or EOI is received. EOS will generate an interrupt when the byte in the Data In Register matches the byte in the EOS register. Otherwise the interrupt will be generated when a true input is detected on EOI.

The GET interrupt bit is used by the microprocessor to detect that DTAS has occurred. It is set by the 8291A when the GET message is received while it is addressed to listen. The TRIG output pin of the 8291A fires when the GET message is received. Thus, the basic operation of device trigger may be started without microprocessor software intervention.

The APT interrupt bit indicates to the processor that a secondary address is available in the CPT register for validation. This interrupt will only occur if Mode 3 addressing is in effect. (Refer to the section on addressing.) In Mode 2, secondary addresses will be recognized automatically on the 8291A. They will be ignored in Mode 1.

The CPT interrupt bit flags the occurrence of an undefined command and of all secondary commands following an undefined command. The Command Pass Through feature is enabled by the B0 bit of Auxiliary Register B. Any message not decoded by the 8291A (not included in the state diagrams in Appendix B) becomes an undefined command. Note that any addressed command is automatically ignored when the 8291A is not addressed.

Undefined commands are read by the CPU from the Command Pass Through register of the 8291A. This register reflects the logic levels present on the data lines at the time it is read. If the CPT feature is enabled, the 8291A will hold off the handshake until this register is read.

An especially useful feature of the 8291A is its ability to generate interrupts from state transitions in the interface functions. In particular, the lower 3 bits of the Interrupt Status 2 Register, if enabled by the corresponding enable bits, will cause an interrupt upon changes in the following states as defined in the IEEE 488 Standard.

Bit 0	ADSC	change in LIDS or TIDS or MJMN
Bit 1	REMC	change in LOCS or REMS
Bit 2	LLOC	change in LWLS or RWLS

The upper 4 bits of the Interrupt Status 2 Register are available to the processor as status bits. Thus, if one of the bits 0–2 generates an interrupt indicating a state change has taken place, the corresponding status bit (bits 3–5) may be read to determine what the new state is. To determine the nature of a change in addressed status (bit 0) the Address Status Register is available to be read. The SPC interrupt (bit 3 in Interrupt Status 2) is set upon exit from SPAS if APRS:STRS:SPAS occurred which indicates that the GPIB controller has read the bus serial poll status byte after the 8291A requested service (asserted SRQ). The SPC interrupt occurs once after the controller reads the status byte if service was requested. The controller may read the status byte later, and the byte will contain the last status the 8291A's CPU wrote to the Serial Poll Mode Register, but the SRQS bit will not be set and no interrupt will be generated. Finally, bit 7 monitors the state of the 8291A INT pin. Logically, it is an OR of all enabled interrupt status bits. One should note that bits 3–6 of the Interrupt Status 2 Register do not generate interrupts, but are available only to be read as status bits by the processor. Bit 7 in Interrupt Status 2 is duplicated in Address 0 Register, and the latter should be used when polling for interrupts to avoid losing one of the interrupts in Interrupt Status 2 Register.

Bits 4 and 5 (DMAI, DMAO) of the Interrupt Mask 2 Register are available to enable direct data transfers between memory and the GPIB; DMAI (DMA in) enables the DREQ (DMA request) pin of the 8291A to be asserted upon the occurrence of BI. Similarly, DMAO (DMA out) enables the DREQ pin to be asserted upon the occurrence of BO. One might note that the DREQ pin may be used as a second interrupt output pin, monitoring BI and/or BO and enabled by DMAI and DMAO. One should note that the DREQ pin is not affected by a read of the Interrupt Status 1 Register. It is reset whenever a byte is written to the Data Out Register or read from the Data In Register.

To ensure that an interrupt status bit will not be cleared without being read, and will not remain uncleared after being read, the 8291A implements a special interrupt handling procedure. When an enabled interrupt bit is set in either of the Interrupt Status Registers, the input of the registers are blocked until the set bit is read and reset by the microprocessor. Thus, potential problems arise when interrupt status changes while the register is being blocked. However, the 8291A stores all new interrupts in a temporary register and transfers them to the appropriate Interrupt Status Register after the interrupt has been reset. This transfer takes place only if the corresponding bits were read as zeroes.

Serial Poll Registers

S8	SRQS	S6	S5	S4	S3	S2	S1

SERIAL POLL STATUS (3R)

S8	rsv	S6	S5	S4	S3	S2	S1

SERIAL POLL MODE (3W)

The Serial Poll Mode Register determines the status byte that the 8291A sends out on the GPIB data lines when it receives the SPE (Serial Poll Enable) message. Bit 6 of this register is reserved for the rsv (request service) local message. Setting this bit to 1 causes the 8291A to assert its $\overline{SRQ}$ line, indicating its need for attention from the controller-in-charge of the GPIB. The other bits of this register are available for sending status information over the GPIB. Sometime after the microprocessor initiates a request for service by setting bit 6, the controller of the GPIB sends the SPE message and then addresses the 8291A to talk. At this point, one byte of status is returned by the 8291A via the Serial Poll Mode Register. After the status byte is read by the controller, rsv is automatically cleared by the 8291A and an SPC interrupt is generated. The CPU may request service again by writing another byte to the Serial Poll Mode Register with the rsv bit set. If the control-

ler performs a serial poll when the rsv bit is clear, the last status byte written will be read, but the $\overline{SRQ}$ line will not be driven by the 8291A and the SRQS bit will be clear in the status byte.

The Serial Poll Status Register is available for reading the status byte in the Serial Poll Mode Register. The processor may check the status of a request for service by polling bit 6 of this register, which corresponds to SRQS (Service Request State). When a Serial Poll is conducted and the controller-in-charge reads the status byte, the SRQS bit is cleared. The $\overline{SRQ}$ line and the rsv bit are tied together.

Address Registers

ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN

ADDRESS STATUS (4R)

INT	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0

ADDRESS 0 (6R)

X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1

ADDRESS 1 (7R)

TO	LO	0	0	0	0	ADM1	ADM0

ADDRESS MODE (4W)

ARS	DT	DL	AD5	AD4	AD3	AD2	AD1

ADDRESS 0/1 (6W)

The Address Mode Register is used to select one of the five modes of addressing available on the 8291A. It determines the way in which the 8291A uses the information in the Address 0 and Address 1 Registers.

—In Mode 1, the contents of the Address 0 Register constitute the "Major" talker/listener address while the Address 1 Register represents the "Minor" talker/listener address. In applications where only one address is needed, the major talker/listener is used, and the minor talker/listener should be disabled. Loading an address via the Address 0/1 Register into Address Registers 0 and 1 enables the major and minor talker/listener functions respectively.

—In Mode 2 the 8291A recognizes two sequential address bytes: a primary followed by a secondary. Both address bytes must be received in order to enable the device to talk or listen. In this manner, Mode 2 addressing implements the extended talker and listener functions as defined in IEEE-488.

To use Mode 2 addressing the primary address must be loaded into the Address 0 Register, and the Secondary Address is placed in the Address 1 Register. With both primary and secondary addresses residing on chip, the 8291A can handle all addressing sequences without processor intervention.

—In Mode 3, the 8291A handles addressing just as it does in Mode 1, except that each Major or Minor primary address must be followed by a secondary address. All secondary addresses must be verified by the microprocessor when Mode 3 is used. When the 8291A is in TPAS or LPAS (talker/listener primary addressed state), and it does not recognize the byte on the DIO lines, an APT interrupt is generated (see section on Interrupt Registers) and the byte is available in the CPT (Command Pass-Through) Register. As part of its interrupt service routine, the microprocessor must read the CPT Register and write one of the following responses to the Auxiliary Mode Register:

1. 07H implies a non-valid secondary address
2. 0FH implies a valid secondary address

Setting the TO bit generates the local ton (talk-only) message and sets the 8291A to a talk-only mode. This mode allows the device to operate as a talker in an interface system without a controller.

Setting the LO bit generates the local lon (listen-only) message and sets the 8291A to a listen-only mode. This mode allows the device to operate as a listener in an interface system without a controller. The above bits may also be used by a controller-in-charge to set itself up for remote command or data communication.

The mode of addressing implemented by the 8291A may be selected by writing one of the following bytes to the Address Mode Register.

Register Contents	Mode
10000000	Enable talk only mode (ton)
01000000	Enable listen only mode (lon)
11000000	The 8291 may talk to itself
00000001	Mode 1, (Primary-Primary)
00000010	Mode 2 (Primary-Secondary)
00000011	Mode 3 (Primary/APT-Primary/APT)

The Address Status Register contains information used by the microprocessor to handle its own addressing. This information includes status bits that monitor the address state of each talker/listener, "ton" and "lon" flags which indicate the talk and listen only states, and an EOI bit which, when set, signifies that the END message came with the last data byte. LPAS and TPAS indicate that the listener

or talker primary address has been received. The microprocessor can use these bits when the secondary address is passed through to determine whether the 8291A is addressed to talk or listen. The LA (listener addressed) bit will be set when the 8219A is in LACS (Listener Active State) or in LADS (Listener Addressed State). Similarly, the TA (Talker Addressed bit) will be set to indicate TACS or TADS, but also to indicate SPAS (Serial Poll Active State). The MJMN bit is used to determine whether the information in the other bits applies to the Major or Minor talker/listener. It is set to "1" when the Minor talker/listener is addressed. It should be noted that only one talker/listener may be active at any one time. Thus, the MJMN bit will indicate which, if either, of the talker/listeners is addressed or active.

The Address 0/1 Register is used for specifying the device's addresses according to the format selected in the Address Mode Register. Five bit addresses may be loaded into the Address 0 and Address 1 Registers by writing into the Address 0/1 Register. The ARS bit is used to select which of these registers the other seven bits will be loaded into. The DT and DL bits may be used to disable the talker or listener function at the address signified by the other five bits. When Mode 1 addressing is used and only one primary address is desired, *both* the talker and the listener should be disabled at the Minor address.

As an example of how the Address 0/1 Register might be used, consider an example where two primary addresses are needed in the device. The Major primary address will be selectable only as a talker and the Minor primary address will be selectable only as a listener. This configuration of the 8291A is formed by the following sequence of writes by the microprocessor.

Operation	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	Data	RS_2–RS_0
1. Select addressing Mode 1	0	1	0	00000001	100
2. Load major address into Address 0 Register with listener function disabled.	0	1	0	001AAAAA	110
3. Load minor address into Address 1 Register with talker function disabled.	0	1	0	110BBBBB	110

At this point, the addresses AAAAA and BBBBB are stored in the Address 0 and Address 1 Registers respectively, and are available to be read by the microprocessor. Thus, it is not necessary to store any address information elsewhere. Also, with the information stored in the Address 0 and Address 1 Registers, processor intervention is not required to recognize addressing by the controller. Only in Mode 3, where secondary addresses are passed through, must the processor intervene in the addressing sequence.

The Address 0 Register contains a copy of bit 7 of the Interrupt Status 2 Register (INT). This is to be used when polling for interrupts. Software should poll register 6 checking for INT (bit 7) to be set. When INT is set, the Interrupt Status Register should be read to determine which interrupt was received.

Command Pass Through Register

CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0

COMMAND PASS THROUGH (5R)

The Command Pass Through Register is used to transfer undefined 8-bit remote message codes from the GPIB to the microprocessor. When the CPT feature is enabled (bit BO in Auxiliary Register B), any message not decoded by the 8291A becomes an undefined command. When Mode 3 addressing is used secondary addresses are also passed through the CPT Register. In either case, the 8291A will hold-off the handshake until the microprocessor reads this register and issues the VSCMD auxiliary command.

The CPT and APT interrupts flag the availability of undefined commands and secondary addresses in the CPT Register. The details of these interrupts are explained in the section on Interrupt Registers.

An added feature of the 8291A is its ability to handle undefined secondary commands following undefined primaries. Thus, the number of available commands for future IEEE-488 definition is increased; one undefined primary command followed by a sequence of as many as 32 secondary commands can be processed. The IEEE-488 Standard does not permit users to define their own commands, but upgrades of the standard are thus provided for.

The recommended use of the 8291A's undefined command capabilities is for a controller-configured Parallel Poll. The PPC message is an undefined primary command typically followed by PPE, and undefined secondary command. For details on this procedure, refer to the section on Parallel Poll Protocol.

Auxiliary Mode Register

CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0

AUX MODE (5W)

CNT0–2:CONTROL BITS

COM0–4:COMMAND BITS

The Auxiliary Mode Register contains a three-bit control field and a five-bit command field. It is used for several purposes on the 8291A:

1. To load "hidden" auxiliary registers on the 8291A.
2. To issue commands from the microprocessor to the 8291A.
3. To preset an internal counter used to generate T1, delay in the Source Handshake function, as defined in IEEE-488.

Table 5 summarizes how these tasks are performed with the Auxiliary Mode Register. Note that the three control bits determine how the five command bits are interpreted.

Table 5

Code		Command
Control Bits	Command Bits	
000	0CCCC	Execute auxiliary command CCCC
001	0DDDD	Preset internal counter to match external clock frequency of DDDD MHz (DDDD binary representation of 1 to 8 MHz)
100	DDDDD	Write DDDDD into auxiliary register A
101	DDDDD	Write DDDDD into auxiliary register B
011	$USP_3P_2P_1$	Enable/disable parallel poll either in response to remote messages (PPC followed by PPE or PPD) or as a local lpe message. (Enable if U = 0, disable if U = 1.)

AUXILIARY COMMANDS

Auxiliary commands are executed by the 8291A whenever 0000CCCC is written into the Auxiliary Mode Register, where CCCC is the 4-bit command code.

0000—Immediate Execute pon: This command resets the 8291A to a power up state (local pon message as defined in IEEE-488).

The following conditions constitute the power up state:

1. All talkers and listeners are disabled.
2. No interrupt status bits are set.

The 8291A is designed to power up in certain states as specified in the IEEE-488 state diagrams. Thus, the following states are in effect in the power up state: SIDS, AIDS, TIDS, LIDS, NPRS, LOCS, and PPIS.

The "0000" pon is an immediate execute command (a pon pulse). It is also used to release the "initialize" state generated by either an external reset pulse or the "0010" Chip Reset command.

0010—Chip Reset (Initialize): This command has the same effect as a pulse applied to the Reset pin. (Refer to the section on Reset Procedure.)

0011—Finish Handshake: This command finishes a handshake that was stopped because of a holdoff on RFD. (Refer to Auxiliary Register A.)

0100—Trigger: A "Group Execute Trigger" is forced by this command. It has the same effect as a GET command issued by the controller-in-charge of the GPIB, but does not cause a GET interrupt.

0101, 1101—Clear/Set rtl: These commands correspond to the local rtl message as defined by the IEEE-488. The 8291A will go into local mode when a Set rtl Auxiliary Command is received if local lockout is not in effect. The 8291A will exit local mode after receiving a Clear rtl Auxiliary Command if the 8291A is addressed to listen.

0110—Send EOI: The EOI line of the 8291A may be asserted with this command. The command causes EOI to go true with the next byte transmitted. The EOI line is then cleared upon completion of the handshake for that byte.

0111, 1111—Non Valid/Valid Secondary Address or Command (VSCMD): This command informs the 8291A that the secondary address received by the microprocessor was valid or invalid (0111 = invalid, 1111 = valid). If Mode 3 addressing is used, the processor must field each extended address and respond to it, or the GPIB will hang up. Note that the COM3 bit is the invalid/valid flag.

The valid (1111) command is also used to tell the 8291A to continue from the command-pass-through-state, or from RFD holdoff on GET, SDC or DCL.

1000—pon: This command puts the 8291A into the pon (power on) state and holds it there. It is similar to a Chip Reset except none of the Auxiliary Mode Registers are cleared. In this state, the 8291A does not participate in any bus activity. An Immediate Execute pon releases the 8291A from the pon state and permits the device to participate in the bus activity again.

0001, 1001—Parallel Poll Flag (local "ist" message): This command sets (1001) or clears (0001) the parallel poll flag. A "1" is sent over the assigned data line (PRR = Parallel Poll Response true) only if the parallel poll flag matches the sense bit from the lpe local message (or indirectly from the PPE message). For a more complete description of the Parallel Poll features and procedures refer to the section on Parallel Poll Protocol.

INTERNAL COUNTER

The internal counter determines the delay time allowed for the setting of data on the DIO lines. This delay time is defined as T_1 in IEEE-488 and appears in the Source Handshake state diagram between the SDYS and STRS. As such, DAV is asserted T_1 after the DIO lines are driven. Consequently, T_1 is a major factor in determining the data transfer rate of the 8291A over the GPIB (T_1 = TWRDV2-TWRD15).

When open-collector transceivers are used for connection to the GPIB, T_1 is defined by IEEE-488 to be 2 μs. By writing 0010DDDD into the Auxiliary Mode Register, the counter is preset to match a f_C MHz clock input, where DDDD is the binary representation of N_F $[1 \leq N_F \leq 8,\ N_F = (DDDD)_2]$. When $N_F = f_C$, a 2 μs T_1 delay will be generated before each DAV asserted.

$$T_{1(\mu s)} = \frac{2N_F}{f_c} + t_{SYNC},\ 1 \leq N_F \leq 8$$

t_{SYNC} is a synchronization error, greater than zero and smaller than the larger of T clock high and T clock low. (For a 50% duty cycle clock, t_{SYNC} is less than half the clock cycle).

If it is necessary that T_1 be different from 2 μs, N_F may be set to a value other than f_C. In this manner, data transfer rates may be programmed for a given system. In small systems, for example, where transfer rates exceeding GPIB specifications are required, one may set $N_F < f_C$ and decrease T_1.

When tri-state transceivers are used, IEEE-488 allows a higher transfer rate (lower T_1). Use of the 8291A with such transceivers is enabled by setting B_2 in Auxilliary Register B. In this case, setting $N_F = f_C$ causes a T_1 delay of 2μs to be generated for the first byte transmitted—all subsequent bytes will have a delay of 500 ns.

$$T_1 \text{ (High Speed) } \mu s = \frac{N_F}{2f_C} + t_{SYNC}$$

Thus, the shortest T_1 is achieved by setting $N_F = 1$ using an 8 MHz clock with a 50% duty cycle clock ($t_{SYNC} < 63$ ns):

$$T_{1(HS)} = \frac{1}{2 \times 8} + 0.063 = 125 \text{ ns max.}$$

AUXILIARY REGISTER A

Auxiliary Register A is a "hidden" 5-bit register which is used to enable some of the 8291A features. Whenever a 100 $A_4A_3A_2A_1A_0$ byte is written into the Auxiliary Register, it is loaded with the data $A_4A_3A_2A_1A_0$. Setting the respective bits to "1" enables the following features.

A_0—RFD Holdoff on all Data: If the 8291A is listening, RFD will not be sent true until the "finish handshake" auxiliary command is issued by the microprocessor. The holdoff will be in effect for each data byte.

A_1—RFD Holdoff on End: This feature enables the holdoff on EOI or EOS (if enabled). However, no hold-off will be in effect on any other data bytes.

A_2—End on EOS Received: Whenever the byte in the Data In Register matches the byte in the EOS Register, the END interrupt bit will be set in the Interrupt Status 1 Register.

A_3—Output EOI on EOS Sent: Any occurrence of data in the Data Out Register matching the EOS Register causes the EOI line to be sent true along with the data.

A_4—EOS Binary Compare: Setting this bit causes the EOS Register to function as a full 8-bit word. When it is not set, the EOS Register is a 7-bit word (for ASCII characters).

If $A_0 = A_1 = 1$, a special "continuous Acceptor Handshake cycling" mode is enabled. This mode should be used only in a controller system configuration, where both the 8291A and the 8292 are used. It provides a continuous cycling through the Acceptor Handshake state diagram, requiring no local messages from the microprocessor; the rdy local message is automatically generated when in ANRS. As such, the 8291A Acceptor Handshake serves as the controller Acceptor Handshake. Thus, the controller cycles through the Acceptor Handshake without delaying the data transfer in progress. When the tcs local message is executed, the 8291A should be taken out of the "continuous AH cycling" mode, the GPIB will hang up in ANRS, and a BI interrupt will be generated to indicate that control may be taken. A

simpler procedure may be used when a "tcs on end of block" is executed; the 8291A may stay in "continuous AH cycling". Upon the end of a block (EOI or EOS received), a holdoff is generated, the GPIB hangs up in ANRS, and control may be taken.

AUXILIARY REGISTER B

Auxiliary Register B is a "hidden" 4-bit register which is used to enable some of the features of the 8291A. Whenever a 101 $B_4B_3B_2B_1B_0$ is written into the Auxiliary Mode Register, it is loaded with the data $B_4B_3B_2B_1B_0$. Setting the respective bits to "1" enables the following features:

B_0—Enable Undefined Command Pass Through: This feature allows any commands not recognized by the 8291A to be handled in software. If enabled, this feature will cause the 8291A to holdoff the handshake when an undefined command is received. The microprocessor must then read the command from the Command Pass Through Register and send the VSCMD auxiliary command. Until the VSCMD command is sent, the handshake holdoff will be in effect.

B_1—Send EOI in SPAS: This bit enables EOI to be sent with the status byte; EOI is sent true in Serial Poll Active State. Otherwise, EOI is sent false in SPAS.

B_2—Enable High Speed Data Transfer: This feature may be enabled when tri-state external transceivers are used. The data transfer rate is limited by T_1 delay time generated in the Source Handshake function, which is defined according to the type of transceivers used. When the "High Speed" feature is enabled, $T_1 = 2$ microseconds is generated for the first byte transmitted after each true to false transition of ATN. For all subsequent bytes, $T_1 =$ 500 ns. Refer to the Internal Counter section for an explanation of T_1 duration as a function of B_2 and of clock frequency.

B_3—Enable Active Low Interrupt: Setting this bit causes the polarity of the INT pin to be reversed, providing an output signal compatible with Intel's MCS-48® Family. Interrupt registers are not affected by this bit.

B_4—Enable RFD Holdoff on GET or DEC: Setting this bit causes RFD to be held false until the "VSCMD" auxiliary command is written after GET, SDC, and DCL commands. This allows the device to hold off the bus until it has completed a clear or trigger similar to an unrecognized command.

PARALLEL POLL PROTOCOL

Writing a 011U$SP_3P_2P_1$ into the Auxiliary Mode Register will enable (U = 0) or disable (U = 1) the 8291A for a parallel poll. When U = 0, this command is the "lpe" (local poll enable) local message as defined in IEEE-488. The "S" bit is the sense in which the 8291A is enabled; only if the Parallel Poll Flag ("ist" local message) matches this bit will the Parallel Poll Response, PPR_N, be sent true (Response = S + ist). The bits $P_3P_2P_1$ specify which of the eight data lines PPR_N will be sent over. Thus, once the 8291A has been configured for Parallel Poll, whenever it senses both EOI and ATN true, it will automatically compare its PP flag with the sense bit and send PPR_N true or false according to the comparison.

If a PP2* implementation is desired, the "lpe" and "ist" local messages are all that are needed. Typically, the user will configure the 8291A for Parallel Poll immediately after initialization. During normal operation the microprocessor will set or clear the Parallel Poll Flag (ist) according to the device's need for service. Consequently the 8291A will be set up to give the proper response to IDY (EOI • ATN) without directly involving the microprocessor.

If a PP1* implementation is desired, the undefined command features of the 8291A must be used. In PP1, the 8291A is indirectly configured for Parallel Poll by the active controller on the GPIB. The sequence at the 8291A being enabled or disabled remotely is as follows:

1. The PPC message is received and is loaded into the Command Pass Through Register as an undefined command. A CPT Interrupt is sent to the microprocessor; the handshake is automatically held off.
2. The microprocessor reads the CPT Register and sends VSCMD to the 8291A, releasing the handshake.
3. Having recieved an undefined primary command, the 8291A is set up to receive an undefined secondary command (the PPE or PPD message). This message is also received into the CPT Register, the handshake is held off, and the CPT interrupt is generated.
4. The microprocessor reads the PPE or PPD message and writes the command into the Auxiliary Mode Register (bit 7 should be cleared first). Finally, the microprocessor sends VSCMD and the handshake is released.

NOTE:
*As defined in IEEE Standard 488.

End of Sequence (EOS) Register

EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

EOS REGISTER

The EOS Register and its features offer an alternative to the "Send EOI" auxiliary command. A seven or eight bit byte (ASCII or binary) may be placed in the register to flag the end of a block or read. The type of EOS byte to be used is selected in Auxiliary Register bit A_4.

If the 8291A is a listener, and the "End on EOS Received" is enabled with bit A_2, then an END interrupt is generated in the Interrupt Status 1 Register whenever the byte in the Data-In Register matches the byte in the EOS Register.

If the 8291A is a talker, and the "Output EOI on EOS Sent" is enabled with bit A_3, then the EOI line is sent true with the next byte whenever the contents of the Data Out Register match the EOS register.

Reset Procedure

The 8291A is reset to an initialization state either by a pulse applied to its Reset pin, or by a reset auxiliary command (02H written into the Auxiliary Command Register). The following conditions are caused by a reset pulse (or local reset command):

1. A "pon" local message as defined by IEEE-488 is held true until the initialization state is released.
2. The Interrupt Status Registers are cleared (not Interrupt Enable Registers).
3. Auxiliary Registers A and B are cleared.
4. The Serial Poll Mode Register is cleared.
5. The Parallel Poll Flag is cleared.
6. The EOI bit in the Address Status Register is cleared.
7. N_F in the Internal Counter is set to 8 MHz. This setting causes the longest possible T_1 delay to be generated in the Source Handshake (16 μs for 1 MHz clock).
8. The rdy local message is sent.

The initialization state is released by an "immediate execute pon" command (00H written into the Auxiliary Command Register).

The suggested initialization sequence is:

1. Apply a reset pulse or send the reset auxiliary command.
2. Set the desired initial conditions by writing into the Interrupt Enable, Serial Poll Mode, Address Mode, Address 0/1, and EOS Registers. Auxiliary Registers A and B, and the internal counter should also be initialized.
3. Send the "immediate execute pon" auxiliary command to release the initialization state.
4. If a PP2 Parallel Poll implementation is to be used the "lpe" local message may be sent, enabling the 8291A for a Parallel Poll Response on an assigned line. (Refer to the section on Parallel Poll Protocol.)

Using DMA

The 8291A may be connected to the Intel® 8237 or 8257 DMA Controllers or the 8089 I/O Processor for DMA operation. The 8237 will be used to refer to any DMA controller. The DREQ pin of the 8291A requests a DMA byte transfer from the 8237. It is set by BO or BI flip flops, enabled by the DMAO and DMAI bits in the Interrupt Enable 2 Register. (After reading , the INT1 register BO and BI interrupts will be cleared but not BO and BI in DREQ equation.)

The $\overline{DACK}$ pin is driven by the 8237 in response to the DMA request. When $\overline{DACK}$ is true (active low) it sets $\overline{CS}$ = RS0 = RS1 = RS2 = 0 such that the $\overline{RD}$ and $\overline{WR}$ signals sent by the 8237 refer to the Data In and Data Out Registers. Also, the DMA request line is reset by $\overline{DACK}$ ($\overline{RD}$ + $\overline{WR}$).

DMA input sequence:

1. A data byte is accepted from the GPIB by the 8291A.
2. A BI interrupt is generated and DREQ is set.
3. $\overline{DACK}$ and $\overline{RD}$ are driven by the 8237, the contents of the Data In Register are transferred to the system bus, and DREQ is reset.
4. The 8291A sends RFD true on the GPIB and proceeds with the Acceptor Handshake protocol.

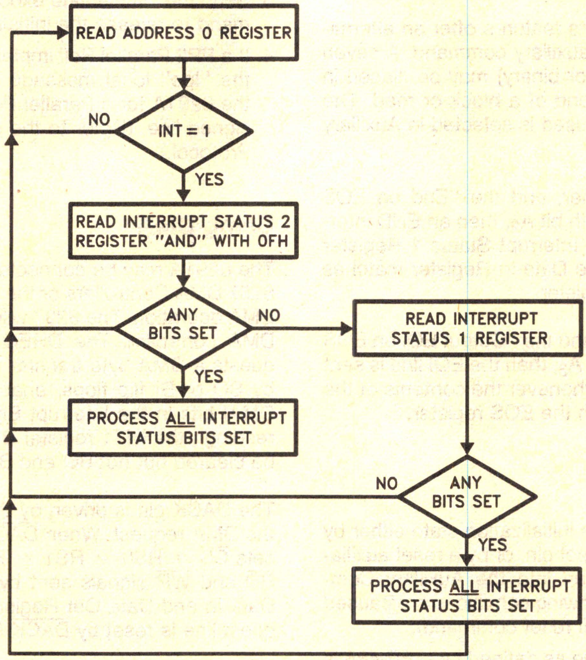
DMA output sequence:

1. A BO interrupt is generated (indicating that a byte should be output) and DREQ is asserted.
2. $\overline{DACK}$ and $\overline{WR}$ are driven by the 8237, a byte is transferred from the MCS bus into the Data Out Register, and DREQ is reset.
3. The 8291A sends DAV true on the GPIB and proceeds with the Source Handshake protocol.

It should be noted that each time the device is addressed (MTA + MLA + ton + lon), the Address Status Register should be read, and the 8237 should be initialized accordingly. (Refer to the 8237 or 8257 Data Sheets.)

Polling the 8291A

If polling is used to determine the 8291A's service needs, the CPU must poll the INT bit in the address 0 register. All relevant interrupt status bits must be enabled during initialization for them to affect the INT status bit. The following flow chart illustrates the recommended polling algorithm.



205248–25

APPLICATION BRIEF

System Configuration

MICROPROCESSOR BUS CONNECTION

The 8291A is 8048/49, 8051, 8080/85, and 8086/88 compatible. The three address pins (RS_0, RS_1, and RS_2) should be connected to the non-multiplexed address bus (for example: A_8, A_9, A_{10}). In case of 8080, any address lines may be used. If the three lowest address bits are used (A_0, A_1, A_2), then they must be demultiplexed first.

EXTERNAL TRANSCEIVERS CONNECTION

The 8293 GPIB Transceiver interfaces the 8291A directly to the IEEE-488 bus. The 8291A and two 8293's can be configured as a talker/listener (see Figure 6) or with the 8292 as a talker/listener/controller (see Figure 7). Absolutely no active or passive external components are required to comply with the complete IEEE-488 electrical specification.

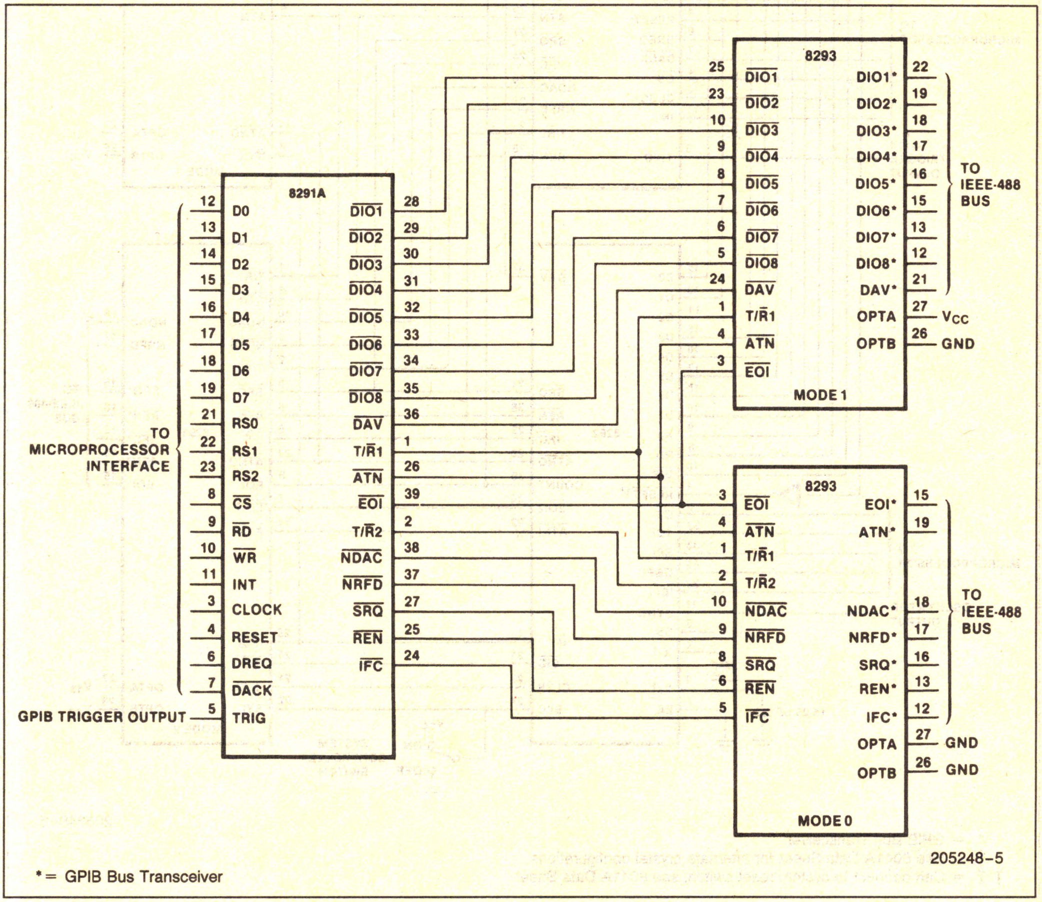
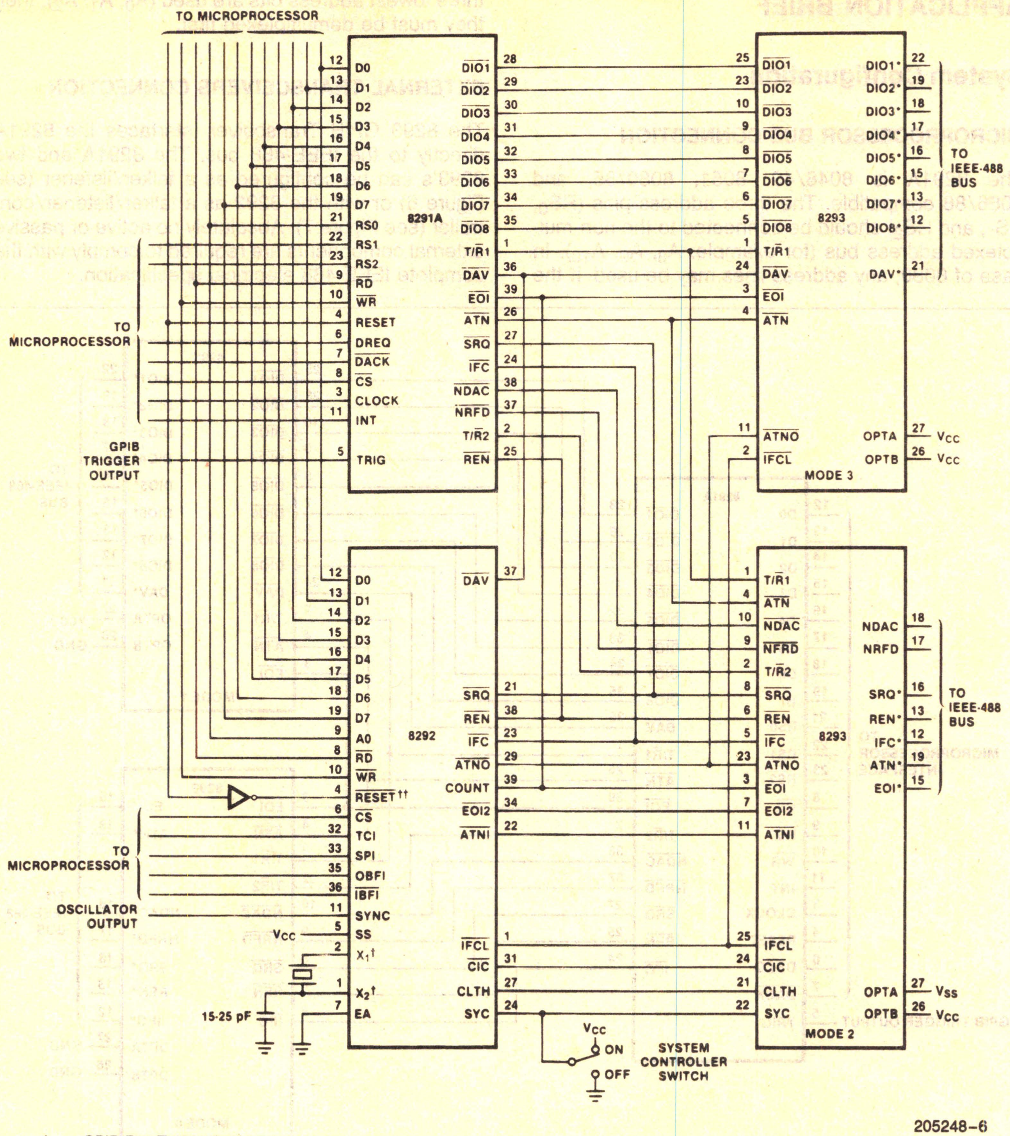


Figure 6. 8291A and 8293 System Configuration



205248-6

* = GPIB Bus Transceiver
† = See 8041A Data Sheet for alternate crystal configurations
†† = Can connect to system reset switch, see 8041A Data Sheet

Figure 7. 8291A, 8292, and 8293 System Configuration

Start-Up Procedures

The following section describes the steps needed to initialize a typical 8291A system implementing a talker/listener interface and an 8291A/8292 system implementing a talker/listener/controller interface.

TALKER/LISTENER SYSTEM

Assume a general system configuration with the following features: (i) Polled system interface; (ii) Mode 1 addressing; (iii) same address for talker and listener; (iv) ASCII carriage return as the end-of-sequence (EOS) character; (v) EOI sent true with the last byte; and, (vi) 8 MHz clock.

Initialization. Initialization is accomplished with the following steps:

1. Pulse the RESET input or write 02H to the Auxiliary Mode Register.
2. Write 00H to the Interrupt Enable Registers 1 and 2. This disables interrupt and DMA.
3. Write 01H to the Address Mode Register to select Mode 1 addressing.
4. Write 28H to the Auxiliary Mode Register. This loads 8H to the Auxiliary Register A matching the 8 MHz clock input to the internal T1 delay counter to generate the delay meeting the IEEE spec.
5. Write the talker/listener address to the Address 0/1 register. The three most significant bits are zero.
6. Write an ASCII carriage return (0DH) to the EOS register.
7. Write 84H to the Auxiliary Mode Register to allow $\overline{\text{EOI}}$ to be sent true when the EOS character is sent.
8. Write 00H to the Auxiliary Mode Register. This writes the "Immediate Execute pon" message and takes the 8291A from the initialization state into the idle state. The 8291A will remain idle until the controller initiates some activity by driving $\overline{\text{ATN}}$ true.

Communication. The local CPU now polls the 8291A to determine which controller command has been received.

The controller addresses the 8291A by driving $\overline{\text{ATN}}$, placing MLA (My Listen Address) on the bus and driving $\overline{\text{DAV}}$. If the lower five bits of the MLA message match the address programmed into the Address 0/1 register, the 8291A is addressed to listen. It would be addressed to talk if the controller sent the MTA message instead of MLA.

The ADSC bit in the Interrupt Status 2 Register indicates that the 8291A has been addressed or unaddressed. The TA and LA bits in the Address Status Register indicate whether the 8291A is talker (TA = 1), listener (LA = 1), both (TA = LA = 1) or unaddressed (TA = LA = 0).

If the 8291A is addressed to listen, the local CPU can read the Data-In Register whenever the BI (Byte In) interrupt occurs in the Interrupt Status 1 Register. If the END bit in the same register is also set, either EOI or a data byte matching the pattern in the EOS register has been received.

In the talker mode, the CPU writes data into the Byte-Out Register on BO (Byte Out) true.

TALKER/LISTENER/CONTROLLER SYSTEM

Combined with the Intel 8292, the 8291A executes a complete IEEE-488-1978 controller function. The 8291A talks and listens via the data and handshake lines ($\overline{\text{NRFD}}$, $\overline{\text{NDAC}}$ and $\overline{\text{DAV}}$). The 8292 controls four of the five bus management lines ($\overline{\text{IFC}}$, $\overline{\text{SRQ}}$, $\overline{\text{ATN}}$ and $\overline{\text{REN}}$). $\overline{\text{EOI}}$, the fifth line, is shared. The 8291A drives and receives $\overline{\text{EOI}}$ when $\overline{\text{EOI}}$ is used as an end-of-block indicator. The 8292 drives $\overline{\text{EOI}}$ along with $\overline{\text{ATN}}$ during a parallel poll command.

Once again, assume a general system configuration with the following features: (i) Polled system interface; (ii) 8292 as the system controller and controller-in-charge; (iii) ASCII carriage return (0DH) as the EOS identifier; (iv) $\overline{\text{EOI}}$ sent with the last character; and, (v) an external buffer (8282) used to monitor the TCI line.

Initialization. In order to send a command across the GPIB, the 8292 has to drive $\overline{\text{ATN}}$, and the 8291A has to drive the data lines. Both devices therefore need initialization.

To initialize the 8292:

1. Pulse the RESET input. The 8292 will initially drive all outputs high. TCI, SPI, OBFI, IBFI and CLTH will then go low. The Interrupt Status, Interrupt Mask, Error Flag, Error Mask and Timeout registers will be cleared. The interrupt counter will be disabled and loaded with 255. The 8292 will then monitor the status of the SYC pin. If high, the 8292 will pulse IFC true for at least 100 μs in compliance with the IEEE-488-1978 standard. It will then take control by asserting $\overline{\text{ATN}}$.

To initialize the 8291A, the following is necessary:

1. Write 00H to Interrupt Enable registers 1 and 2. This disables interrupt and DMA.

2. With the 8292 as the controller-in-charge, it is impossible to address the 8292 via the GPIB. Therefore, the ton or lon modes of the 8291A must be used. To send commands, set the 8291A in the ton mode by writing 80H to the Address Mode Register.
3. Write 26H to the Auxiliary Mode Register to match the T1 data settling time to the 6 MHz clock input.
4. Write an ASCII carriage return (0DH) to the EOS Register.
5. Write 84H to the Auxiliary Mode Register in order to enable "Output EOI on EOS sent" and thus send EOI with the last character.
6. Write 00H—Immediate Execute pon—to the Auxiliary Mode Register to put the 8291A in the idle state.

Communication. Since the 8291A is in the ton mode, a BO interrupt is generated as soon as the immediate Execute pon command is written. The CPU writes the command into the Data Out Register, and repeats it on BO becoming true for as many commands as necessary. $\overline{\text{ATN}}$ remains continuously true unless the GTSB (Go To Standby) command is sent to the 8292.

$\overline{\text{ATN}}$ has to be false in order to send data rather than commands from the controller. To do this, the following steps are needed:

1. Enable the TCI interrupt if not already enabled.
2. Wait for IBF (Input Buffer Full) in the 8292 Interrupt Status Register to be reset.
3. Write the GTSB (F6H) command to the 8292 Command Field Register.
4. Read the 8282 and wait for TCI to be true.
5. Write the ton (80H) and pon (00H) command to the 8291A Address Mode Register and Auxiliary Mode Registers respectively.
6. Wait for the BO interrupt to be set in the 8291A.
7. Write the data to the 8291A Data-Out Register.

Identically, the user could command the controller to listen rather than talk. To do that, write lon (40H) instead of ton into the Address Mode Register. Then wait for BI rather than BO to go true. Read the data Register.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias ...... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground .... −0.5V to +7V
Power Dissipation .... 0.65 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $V_{CC} = 5V \pm 10\%$, $T_A = 0°C$ to 70°C (Commercial)

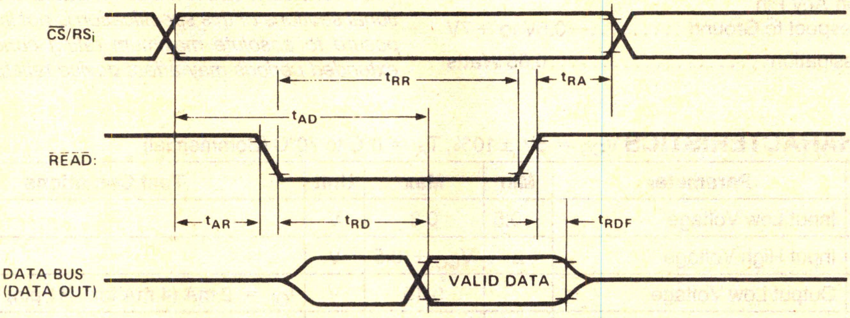
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2	$V_{CC} + 0.5$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL} = 2$ mA (4 mA for TR1 pin)
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -400\ \mu A$ ($-150\ \mu A$ for SRQ pin)
$V_{OH\text{-}INT}$	Interrupt Output High Voltage	2.4 3.5		V V	$I_{OH} = -400\ \mu A$ $I_{OH} = -50\ \mu A$
I_{IL}	Input Leakage		10	μA	$V_{IN} = 0V$ to V_{CC}
I_{OFL}	Output Leakage Current		±10	μA	$V_{OUT} = 0.45V$, V_{CC}
I_{CC}	V_{CC} Supply Current		120	mA	$T_A = 0°C$

A.C. CHARACTERISTICS $V_{CC} = 5V \pm 10\%$, $T_A = 0°C$ to 70°C (Commercial)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AR}	Address Stable Before $\overline{READ}$	0		ns	
t_{RA}	Address Hold After $\overline{READ}$	0		ns	
t_{RR}	$\overline{READ}$ Width	140		ns	
t_{AD}	Address Stable to Data Valid		250	ns	
t_{RD}	$\overline{READ}$ to Data Valid		100	ns	
t_{RDF}	Data Float After $\overline{READ}$	0	60	ns	
t_{AW}	Address Stable Before $\overline{WRITE}$	0		ns	
t_{WA}	Address Hold After $\overline{WRITE}$	0			
t_{WW}	$\overline{WRITE}$ Width	170		ns	
t_{DW}	Data Set Up Time to the Trailing Edge of WRITE	130		ns	
t_{WD}	Data Hold Time After $\overline{WRITE}$	0		ns	
t_{DKDR4}	$\overline{RD}\downarrow$ or $\overline{WR}\downarrow$ to DREQ$\downarrow$		130	ns	
t_{DKDA6}	$\overline{RD}\downarrow$ to Valid Data (D_0–D_7)		200	ns	$\overline{DACK}\downarrow$ to $\overline{RD}\downarrow$ $0 \le t \le 50$ ns

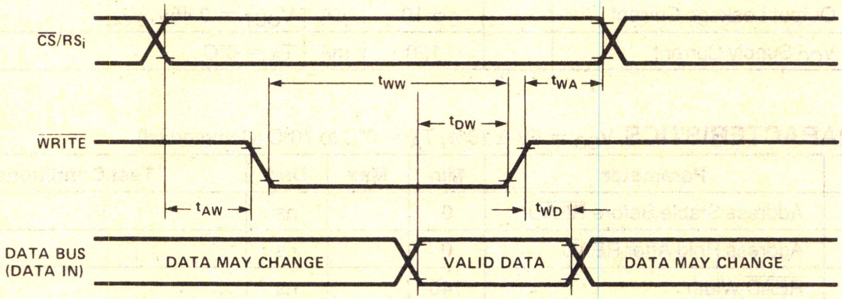
WAVEFORMS

READ



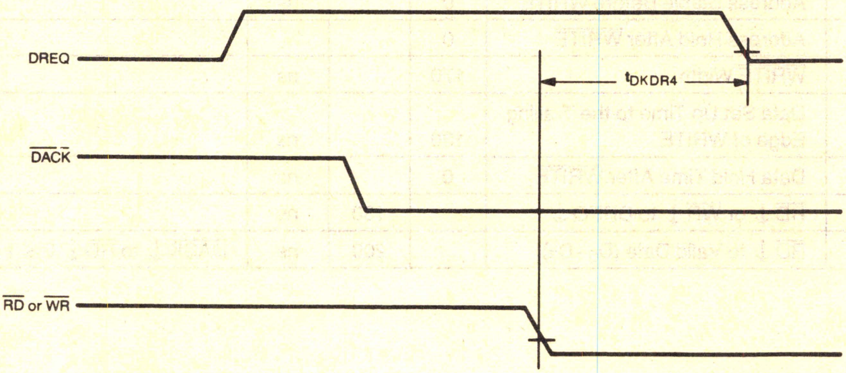
205248-7

WRITE



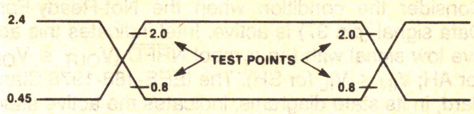
205248-8

DMA



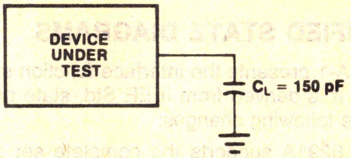
205248-9

A.C. TIMING MEASUREMENT POINTS AND LOAD CONDITIONS



205248–10

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".



205248–11

GPIB TIMINGS(1)

Symbol	Parameter	Max	Units	Test Conditions
TEOT13(2)	$\overline{EOI}$ ↓ to TR1 ↑	135	ns	PPSS, ATN = 0.45V
TEOD16	$\overline{EOI}$ ↓ to $\overline{DIO}$ Valid	155	ns	PPSS, ATN = 0.45V
TEOT12	$\overline{EOI}$ ↑ to TR1 ↓	155	ns	PPSS, ATN = 0.45V
TATND4	$\overline{ATN}$ ↓ to $\overline{NDAC}$ ↓	155	ns	TACS, AIDS
TATT14	$\overline{ATN}$ ↓ to TR1 ↓	155	ns	TACS, AIDS
TATT24	$\overline{ATN}$ ↓ to TR2 ↓	155	ns	TACS, AIDS
TDVND3-C	$\overline{DAV}$ ↓ to $\overline{NDAC}$ ↑	650	ns	AH, CACS
TNDDV1	$\overline{NDAC}$ ↑ to $\overline{DAV}$ ↑	350	ns	SH, STRS
TNRDR1	$\overline{NRFD}$ ↑ to $\overline{DREQ}$ ↑	400	ns	SH
TDVDR3	$\overline{DAV}$ ↓ to $\overline{DREQ}$ ↑	600	ns	AH, LACS, ATN = 2.4V
TDVND2-C	$\overline{DAV}$ ↑ to $\overline{NDAC}$ ↓	350	ns	AH, LACS
TDVNR1-C	$\overline{DAV}$ ↑ to $\overline{NRFD}$ ↑	350	ns	AH, LACS, rdy = True
TRDNR3	$\overline{RD}$ ↓ to $\overline{NRFD}$ ↑	500	ns	AH, LACS
TWRD15	$\overline{WR}$ ↑ to $\overline{DIO}$ Valid	280	ns	SH, TACS, RS = 0.4V
TWREO5	$\overline{WR}$ ↑ to $\overline{EOI}$ Valid	350	ns	SH, TACS
TWRDV2	$\overline{WR}$ ↑ to $\overline{DAV}$ ↓	830 + t_{SYNC}	ns	High Speed Transfers Enabled, $N_F = f_C$, $t_{SYNC} = \frac{1}{2} \bullet f_C$

NOTES:
1. All GPIB timings are at the pins of the 8291A.
2. The last number in the symbol for any GPIB timing parameter is chosen according to the transition directions of the reference signals. The following table describes the numbering scheme.

↑ to ↑	1
↑ to ↓	2
↓ to ↑	3
↓ to ↓	4
↑ to VALID	5
↓ to VALID	6

APPENDIX A

MODIFIED STATE DIAGRAMS

Figure A-1 presents the interface function state diagrams. It is derived from IEEE Std. state diagrams, with the following changes:

A. The 8291A supports the complete set of IEEE-488 interface functions except for the controller. These include: SH1, AH1, T5, TE5, L3, LE3, SR1, RL1, PP1, DC1, DT1, and C0.

B. Addressing modes included in T, L state diagrams.

Note that in Mode 3, MSA, OSA are generated only after secondary address validity check by the microprocessor (APT interrupt).

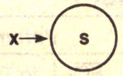
C. In these modified state diagrams, the IEEE-488-1978 convention of negative (low true) logic is followed. This should not be confused with the Intel pin- and signal-naming convention based on positive logic. Thus, while the state diagrams below carry low true logic, the signals described elsewhere in this data sheet are consistent with Intel notation and are based on positive logic.

Level	Logic	Convention	
		IEEE-488	Intel
0	T	DAV	$\overline{DAV}$
1	F	$\overline{DAV}$	DAV
0	T	NDAC	$\overline{NDAC}$
1	F	$\overline{NDAC}$	NDAC
0	T	NRFD	$\overline{NRFD}$
1	F	$\overline{NRFD}$	NRFD

Consider the condition when the Not-Ready-For-Data signal (pin 37) is active. Intel indicates this active low signal with the symbol $\overline{NRFD}$ ($V_{OUT} \leq V_{OL}$ for AH; $V_{IN} \leq V_{IL}$ for SH). The IEEE-488-1978 Standard, in its state diagrams, indicates the active state of this signal (True condition) with NRFD.

D. All remote multiline messages decoded are conditioned by ACDS. The multiplication by ACDS is not drawn to simplify the diagrams.

E. The symbol



indicates:

1. When event X occurs, the function returns to state S.
2. X overrides any other transition condition in the function.

Statement 2 simplifies the diagram, avoiding the explicit use of X to condition all transitions from S to other states.

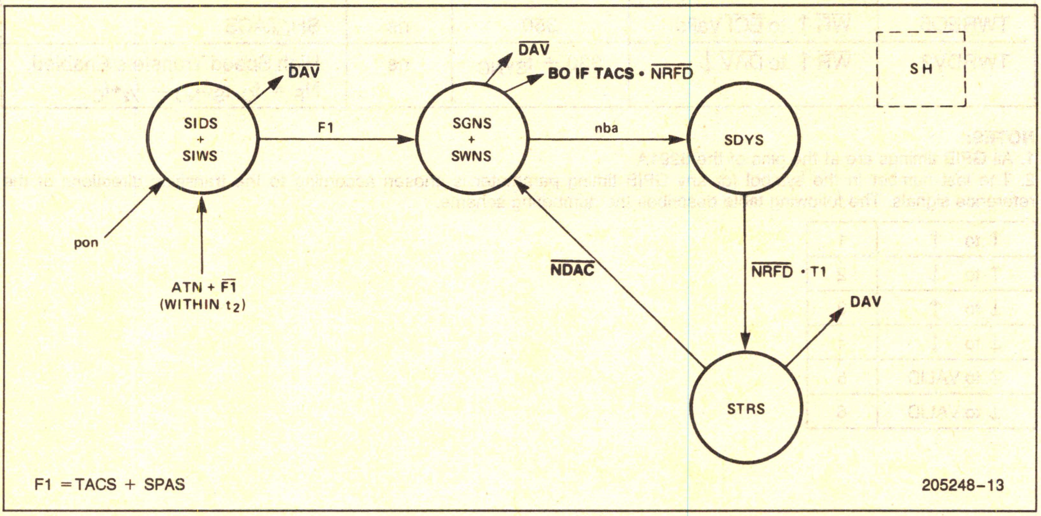
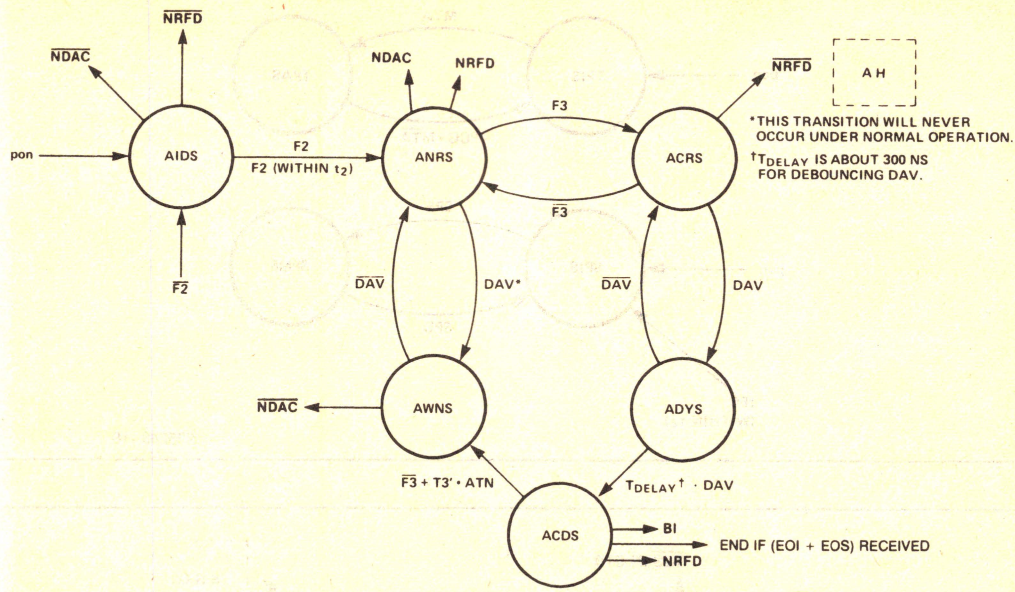
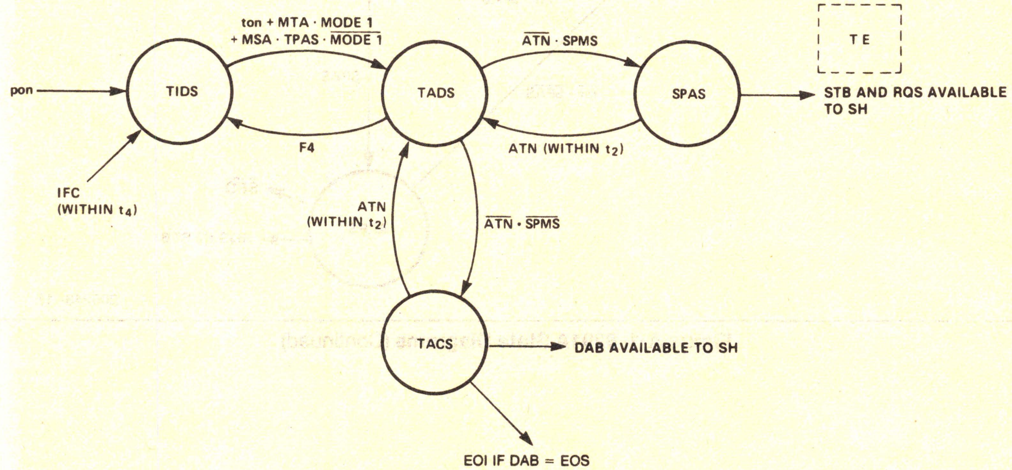


Figure A-1. 8291A State Diagrams



F2 = ATN + LACS + LADS
F3 = ATN + rdy
T3' = T3 • $\overline{\text{CPT}}$ • $\overline{\text{APT}}$

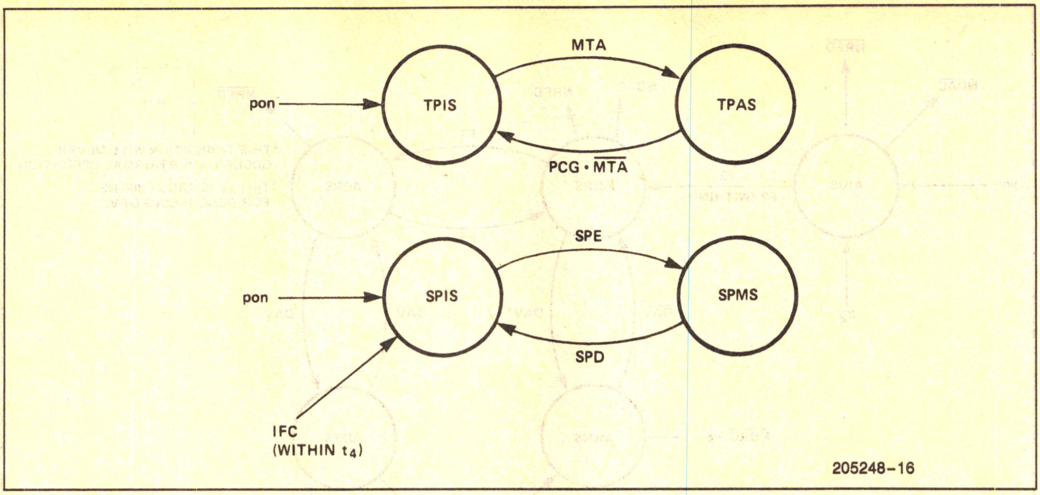
205248-14



F4 = OTA + (OSA • TPAS + MSA • LPAS) •
$\overline{\text{MODE 1}}$ + MLA • MODE 1

205248-15

Figure A-1. 8291A State Diagrams (Continued)



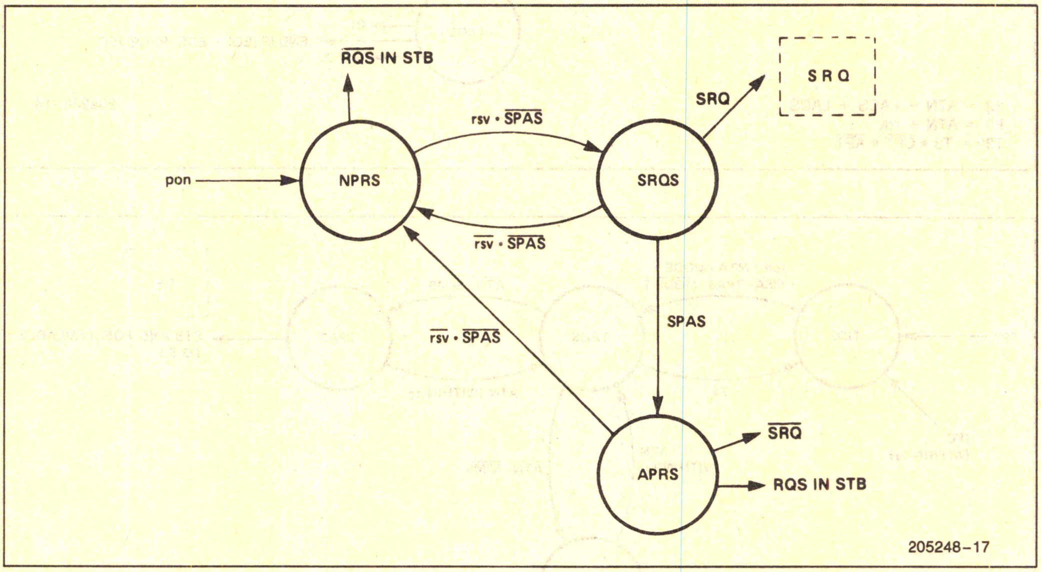
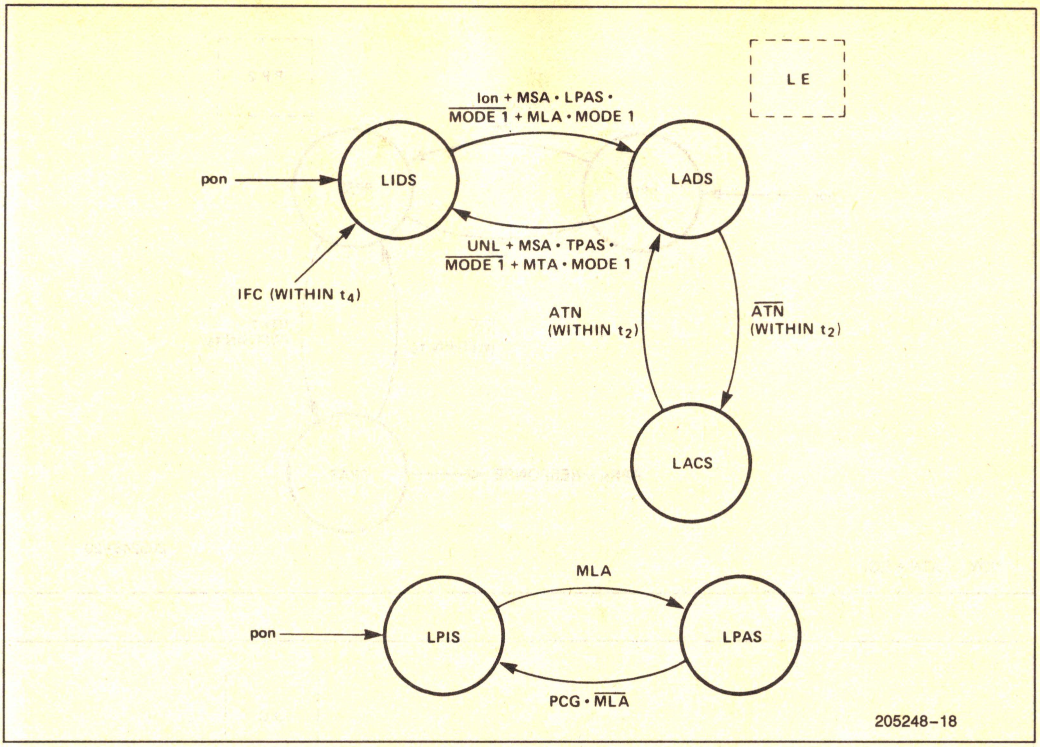


Figure A-1. 8291A State Diagrams (Continued)



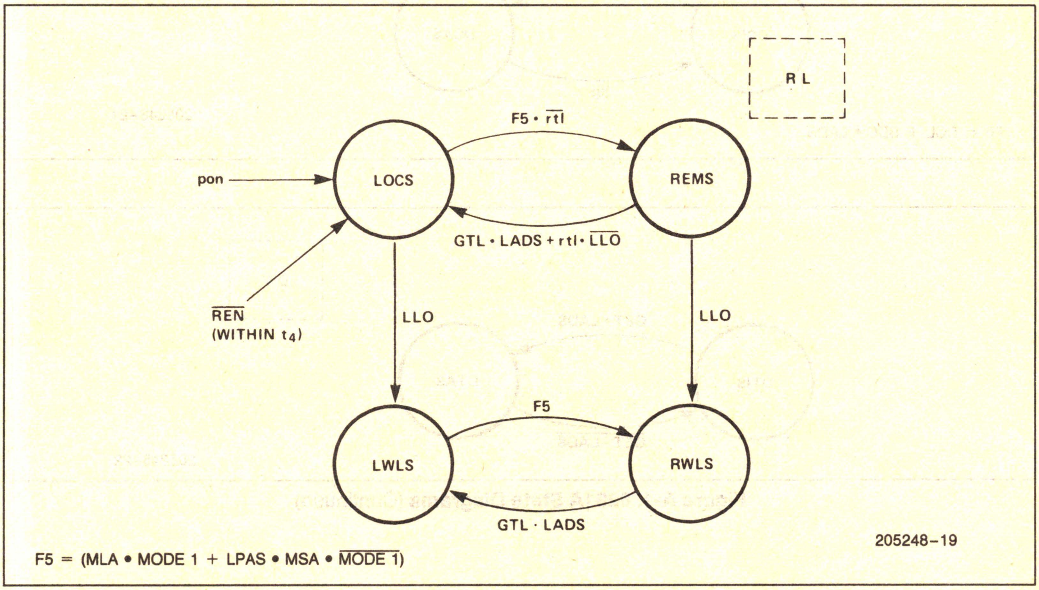
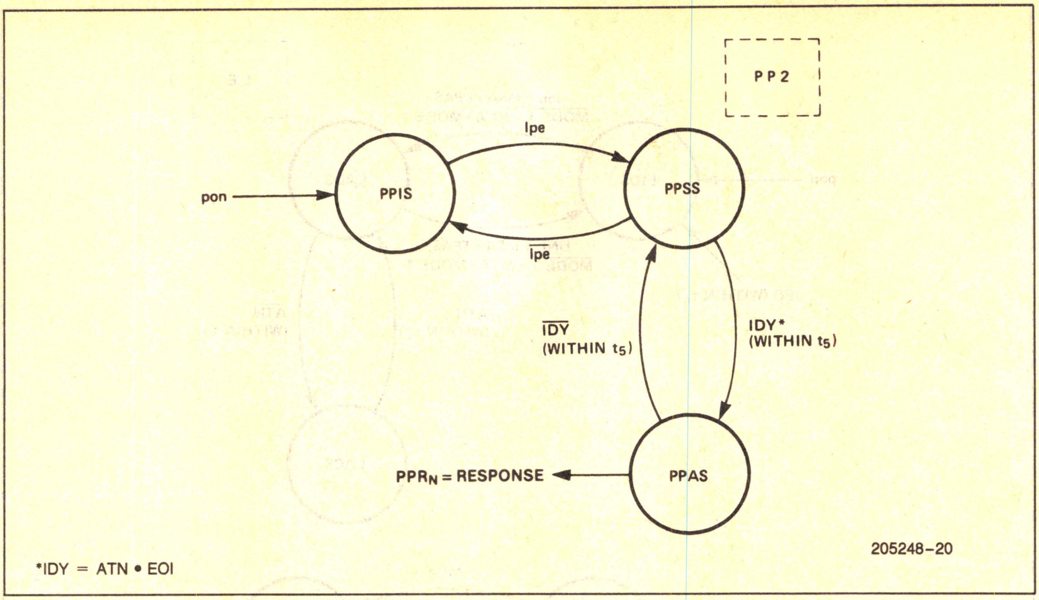
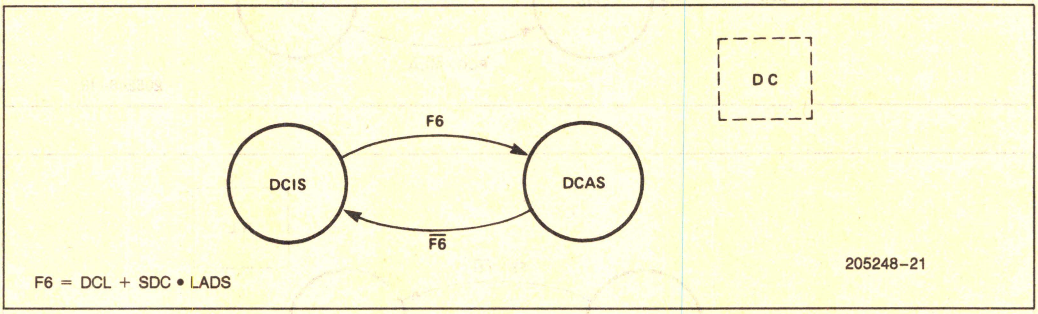


Figure A-1. 8291A State Diagrams (Continued)





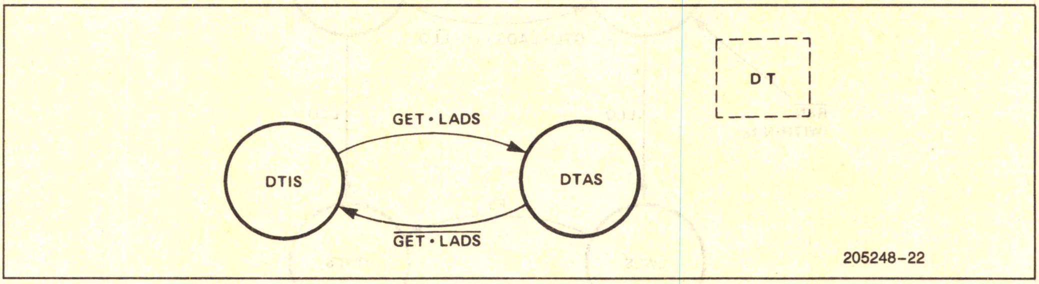


Figure A-1. 8291A State Diagrams (Continued)

APPENDIX B

Table B-1. IEEE 488 Time Values

Time Value Identifier(1)	Function (Applies to)	Description	Value
T_1	SH	Settling Time for Multiline Messages	$\geq 2\ \mu s$(2)
t_2	LC, $\overline{IC}$, SH, AH, T, L	Response to ATN	≤ 200 ns
T_3	AH	Interface Message Accept Time(3)	> 0(4)
t_4	T, TE, L, LE, C, CE	Response to IFC or REN False	$< 100\ \mu s$
t_5	PP	Response to ATN + EOI	≤ 200 ns
T_6	C	Parallel Poll Execution Time	$\geq 2\ \mu s$
T_7	C	Controller Delay to Allow Current Talker to see ATN Message	≥ 500 ns
T_8	C	Length of IFC or REN False	$> 100\ \mu s$
T_9	C	Delay for EOI(5)	$\geq 1.5\ \mu s$(6)

NOTES:

1. Time values specified by a lower case t indicate the maximum time allowed to make a state transition. Time values specified by an upper case T indicate the minimum time that a function must remain in a state before exiting.
2. If three-state drivers are used on the $\overline{DIO}$, $\overline{DAV}$, and $\overline{EOI}$ lines, T_1 may be:
   1. $\geq$ 1100 ns.
   2. Or $\geq$ 700 ns if it is known that within the controller ATN is driven by a three-state driver.
   3. Or $\geq$ 500 ns for all subsequent bytes following the first sent after each false transition of ATN (the first byte must be sent in accordance with (1) or (2).
   4. Or $\geq$ 350 ns for all subsequent bytes following the first sent after each false transition of ATN under conditions specified in Section 5.2.3 and warning note. See IEEE Standard 488.
3. Time required for interface functions to accept, not necessarily respond to interface messages.
4. Implementation dependent.
5. Delay required for $\overline{EOI}$, $\overline{NDAC}$, and $\overline{NRFD}$ signal lines to indicate valid states.
6. $\geq$ 600 ns for three-state drivers.

APPENDIX C
THE THREE-WIRE HANDSHAKE

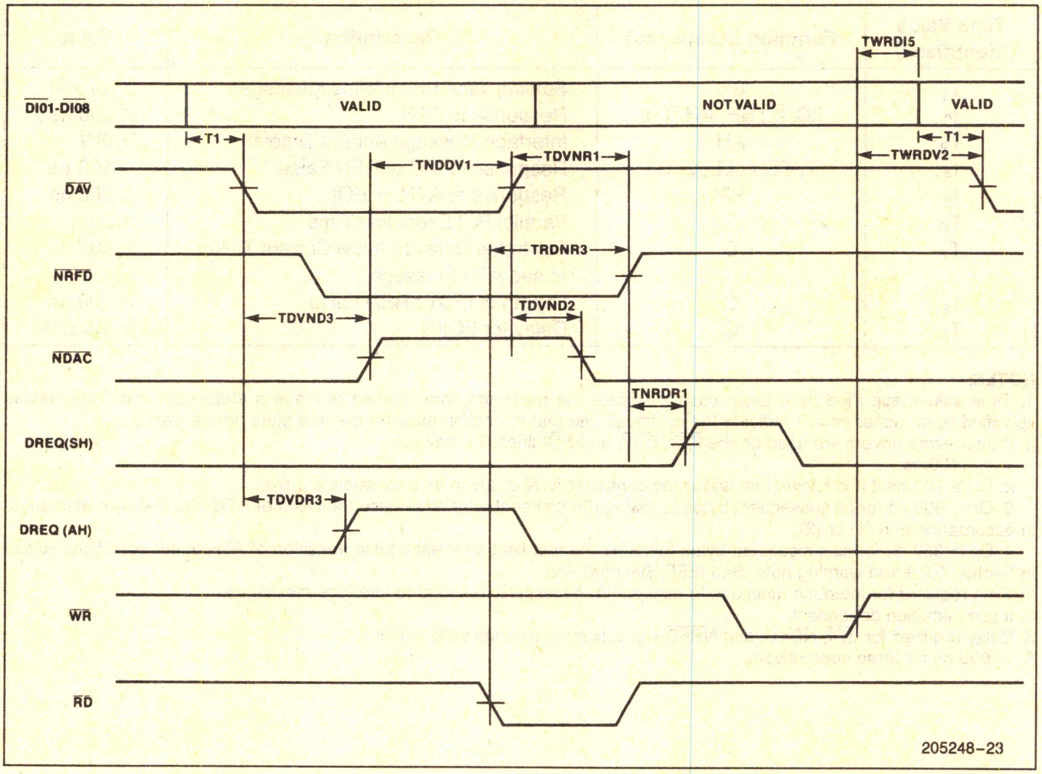


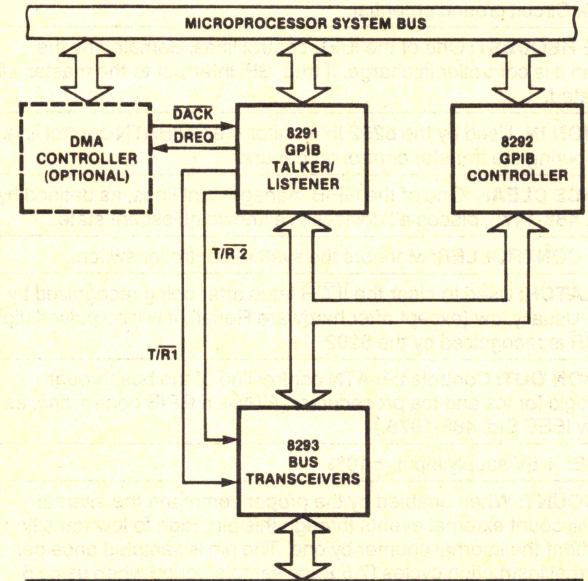
Figure C-1. 3-Wire Handshake Timing at 8291A

intel®

8292
GPIB CONTROLLER

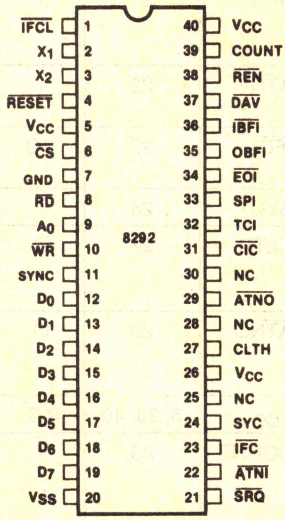
- **Complete IEEE Standard 488 Controller Function**
- **Interface Clear (IFC) Sending Capability Allows Seizure of Bus Control and/or Initialization of the Bus**
- **Responds to Service Requests (SRQ)**
- **Sends Remote Enable (REN), Allowing Instruments to Switch to Remote Control**
- **Complete Implementation of Transfer Control Protocol**
- **Synchronous Control Seizure Prevents the Destruction of Any Data Transmission in Progress**
- **Connects with the 8291 to Form a Complete IEEE Standard 488 Interface Talker/Listener/Controller**

The 8292 GPIB Controller is a microprocessor-controlled chip designed to function with the 8291 GPIB Talker/Listener to implement the full IEEE Standard 488 controller function, including transfer control protocol. The 8292 is a preprogrammed Intel® 8041A.



205250-1

Figure 1. 8291, 8292 Block Diagram



205250-2

Figure 2. Pin Configuration

November 1986
Order Number: 205250-002

Table 1. Pin Description

Symbol	Pin Number	Type	Name and Function
$\overline{IFCL}$	1	I	**IFC RECEIVED (LATCHED):** The 8292 monitors the IFC Line (when not system controller) through this pin.
X_1, X_2	2, 3	I	**CRYSTAL INPUTS:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
$\overline{RESET}$	4	I	**RESET:** Used to initialize the chip to a known state during power on.
$\overline{CS}$	6	I	**CHIP SELECT INPUT:** Used to select the 8292 from other devices on the common data bus.
$\overline{RD}$	8	I	**READ ENABLE:** Allows the master CPU to read from the 8292.
A_0	9	I	**ADDRESS LINE:** Used to select between the data bus and the status register during read operations and to distinguish between data and commands written into the 8292 during write operations.
$\overline{WR}$	10	I	**WRITE ENABLE:** Allows the master CPU to write to the 8292.
SYNC	11	O	**SYNC:** 8041A instruction cycle synchronization signal; it is an output clock with a frequency of XTAL ÷ 15.
D_0–D_7	12–19	I/O	**DATA:** 8 bidirectional lines used for communication between the central processor and the 8292's data bus buffers and status register.
V_{SS}	7, 20	P.S.	**GROUND:** Circuit ground potential.
$\overline{SRQ}$	21	I	**SERVICE REQUEST:** One of the IEEE control lines. Sampled by the 8292 when it is controller in charge. If true, SPI interrupt to the master will be generated.
$\overline{ATNI}$	22	I	**ATTENTION IN:** Used by the 8292 to monitor the GPIBATN control line. If is used during the transfer control procedure.
$\overline{IFC}$	23	I/O	**INTERFACE CLEAR:** One of the GPIB management lines, as defined by IEEE Std. 488-1978, places all devices in a known quiescent state.
SYC	24	I	**SYSTEM CONTROLLER:** Monitors the system controller switch.
CLTH	27	O	**CLEAR LATCH:** Used to clear the $\overline{IFCR}$ latch after being recognized by the 8292. Usually low (except after hardware Reset), it will be pulsed high when $\overline{IFCR}$ is recognized by the 8292.
$\overline{ATNO}$	29	O	**ATTENTION OUT:** Controls the ATN control line of the bus through external logic for tcs and tca procedures. (ATN is a GPIB control line, as defined by IEEE Std. 488-1978.)
V_{CC}	5, 26, 40	P.S.	**VOLTAGE:** +5V supply input ±10%.
COUNT	39	I	**EVENT COUNT:** When enabled by the proper command the internal counter will count external events through this pin. High to low transition will increment the internal counter by one. The pin is sampled once per three internal instruction cycles (7.5 μsec sample period when using 5 MHz XTAL). It can be used for byte counting when connected to NDAC, or for block counting when connected to the EOI.
$\overline{REN}$	38	O	**REMOTE ENABLE:** The Remote Enable bus signal selects remote or local control of the device on the bus. A GPIB bus signal selects remote or local control of the device on the bus. A GPIB bus management line, as defined by IEEE Std. 488-1978.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{DAV}$	37	I/O	**DATA VALID:** Used during parallel poll to force the 8291 to accept the parallel poll status bit. It is also used during the tcs procedure.
$\overline{IBFI}$	36	O	**INPUT BUFFER NOT FULL:** Used to interrupt the central processor while the input buffer of the 8292 is empty. This feature is enabled and disabled by the interrupt mask register.
OBFI	36	O	**OUTPUT BUFFER FULL:** Used as an interrupt to the central processor while the output buffer of the 8292 is full. The feature can be enabled and disabled by the interrupt mask register.
$\overline{EOI2}$	34	I/O	**END OR IDENTIFY:** One of the GPIB management lines, as defined by IEEE Std. 488-1978. Used with ATN as Identify Message during parallel poll.
SPI	33	O	**SPECIAL INTERRUPT:** Used as an interrupt on events not initiated by the central processor.
TCI	32	O	**TASK COMPLETE INTERRUPT:** Interrupt to the control processor used to indicate that the task requested was completed by the 8292 and the information requested is ready in the data bus buffer.
CIC	31	O	**CONTROLLER IN CHARGE:** Controls the S/R input of the SRQ bus transceiver. It can also be used to indicate that the 8292 is in charge of the GPIB bus.

FUNCTIONAL DESCRIPTION

The 8292 is an Intel 8041A which has been programmed as a GPIB Controller Interface element. It is used with the 8291 GPIB Talker/Listener and two 8293 GPIB Transceivers to form a complete IEEE-488 Bus Interface for a microprocessor. The electrical interface is performed by the transceivers, data transfer is done by the talker/listener, and control of the bus is done by the 8292. Figure 3 is a typical controller interface using Intel's GPIB peripherals.

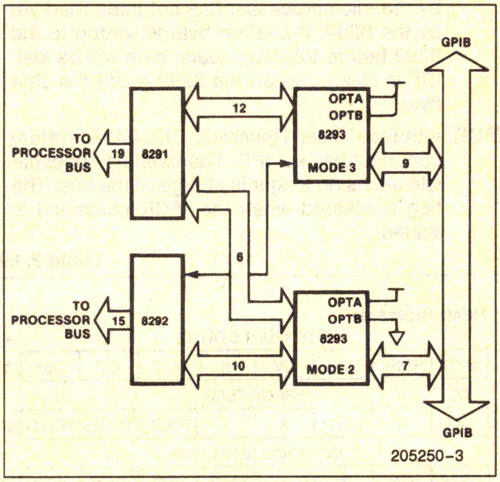


Figure 3. Talker/Listener/Controller Configuration

The internal RAM in the 8041A is used as a special purpose register bank for the 8292. Most of these registers (except for the interrupt flag) can be accessed through commands to the 8292. Table 2 identifies the registers used by the 8292 and how they are accessed.

Interrupt Status Register

D_7							D_0
SYC	ERR	SRQ	EV	X	IFCR	IBF	OBF

The 8292 can be configured to interrupt the microprocessor on one of several conditions. Upon receipt of the interrupt the microprocessor must read the 8292 interrupt status register to determine which event caused the interrupt, and then the appropriate subroutine can be performed. The interrupt status register is read with A_0 high. With the exception of OBF and IBF, these interrupts are enabled or disabled by the SPI interrupt mask. OBF and IBF have their own bits in the interrupt mask (OBFI and $\overline{\text{IBFI}}$).

OBF Output Buffer Full. A byte is waiting to be read by the microprocessor. This flag is cleared when the output data bus buffer is read.

IBF Input Buffer Full. The byte previously written by the microprocessor has not been read yet by the 8292. If another byte is written to the 8292 before this flag clears, data will be lost. IBF is cleared when the 8292 reads the data byte.

IFCR Interface Clear Received. The GPIB system controller has set IFC. The 8292 has become idle and is no longer is charge of the bus. The flag is cleared when the IACK command is issued.

EV Event Counter Interrupt. The requested number of blocks of data byte has been transferred. The EV interrupt flag is cleared by the IACK command.

SRQ Service Request. Notified the 8292 that a service request (SRQ) message has been received. It is cleared by the IACK command.

ERR Error occurred. The type of error can be determined by reading the error status register. This interrupt flag is cleared by the IACK command.

SYC System Controller Switch Change. Notifies the processor that the state of the system controller switch has changed. The actual state is contained in the GPIB Status Register. This flag is cleared by the IACK command.

Interrupt Mask Register

D_7							D_0
1	SPI	TCI	SYC	OBFI	$\overline{\text{IBFI}}$	0	SRQ

The Interrupt Mask Register is used to enable features and to mask the SPI and TCI interrupts. The flags in the Interrupt Status Register will be active even when masked out. The Interrupt Mask Register is written when A_0 is low and reset by the RINM command. When the register is read, D_1 and D_7 are undefined. An interrupt is enabled by setting the corresponding register bit.

SRQ Enable interrupts on SRQ received.

$\overline{\text{IBFI}}$ Enable interrupts on input buffer empty.

OBFI Enable interrupts on output buffer full.

Table 2. 8292 Registers

READ FROM 8292

Register	D_7							D_0	A_0
INTERRUPT STATUS	SYC	ERR	SRQ	EV	X	IFCR	IBF	OBF	1
ERROR FLAG	X	X	USER	X	X	$TOUT_3$	$TOUT_2$	$TOUT_1$	0*
CONTROLLER STATUS	CSBS	CA	X	X	SYCS	IFC	REN	SRQ	0*
GPIB (BUS) STATUS	REN	DAV	EOI	X	SYC	IFC	ANTI	SRQ	0*
EVENT COUNTER STATUS	D	D	D	D	D	D	D	D	0*
TIME OUT STATUS	D	D	D	D	D	D	D	D	0*

WRITE TO 8292

Register	D_7							D_0	A_0
INTERRUPT MASK	1	SPI	TCI	SYC	OBFI	$\overline{\text{IBFI}}$	0	SRQ	0
ERROR MASK	0	0	USER	0	0	$TOUT_3$	$TOUT_2$	$TOUT_1$	0
COMMAND FIELD	1	1	1	OP	C	C	C	C	1
EVENT COUNTER	D	D	D	D	D	D	D	D	0*
TIME OUT	D	D	D	D	D	D	D	D	0*

NOTE: These registers are accessed by a special utility command, see page 7.

SYC Enable interrupts on a change in the system controller switch.

TCI Enable interrupts on the task completed.

SPI Enable interrupts on special events.

NOTE:
The event counter is enabled by the GSEC command, the error interrupt is enabled by the error mask register, and IFC cannot be masked (it will always cause an interrupt).

Controller Status Register

CSBS	CA	X	X	SYCS	IFC	REN	SRQ

D_7 D_0

The Controller Status Register is used to determine the status of the controller function. This register is accessed by the RCST command.

SRQ Service Request line active (CSRS).

REN Sending Remote Enable.

IFC Sending or receiving interface clear.

SYCS System Controller Switch Status (SACS).

CA Controller Active (CACS + CAWS + CSWS).

CSBS Controller Stand-by State (CSBS, CA) = (0,0)—Controller Idle.

GPIB Bus Status Register

REN	DAV	EOI	X	SYC	IFC	ATNI	SRQ

D_7 D_0

This register contains GPIB bus status information. It can be used by the microprocessor to monitor and manage the bus. The GPIB Bus Register can be read using the RBST command.

Each of these status bits reflect the current status of the corresponding pin on the 8292.

SRQ Service Request

ATNI Attention In

IFC Interface Clear

SYC System Controller Switch

EOI End or Identify

DAV Data Valid

REN Remote Enable

Event Counter Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

The Event Counter Register contains the initial value for the event counter. The counter can count pulses on pin 39 of the 8292 (COUNT). It can be connected to EOI or NDAC to count blocks or bytes respectively during standby state. A count of zero equals 256. This register cannot be read, and is written using the WEVC command.

Event Counter Status Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

This register contains the current value in the event counter. The event counter counts back from the initial value stored in the Event Counter Register to zero and then generates an Event Counter Interrupt. This register cannot be written and can be read using a REVC command.

Time Out Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

The Time Out Register is used to store the time used for the time out error function. See the individual timeouts (TOUT1, 2, 3) to determine the units of this counter. This Time Out Register cannot be read, and it is written with the WTOUT command.

Time Out Status Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

This register contains the current value in the time out counter. The time out counter decrements from the original value stored in the Time Out Register. When zero is reached, the appropriate error interrupt is generated. If the register is read while none of the time out functions are active, the register will contain the last value reached the last time a function was active. The Time Out Status Register cannot be written, and it is read with RTOUT command.

Error Flag Register

X	X	USER	X	X	$TOUT_3$	$TOUT_2$	$TOUT_1$

Four errors are flagged by the 8292 with a bit in the Error Flag Register. Each of these errors can be masked by the Error Mask Register. The Error Flag Register cannot be written, and it is read by the IACK command when the error flag in the Interrupt Status Register is set.

TOUT1 Time Out Error 1 occurs when the current controller has not stopped sending ATN after receiving the TCT message for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 1800 t_{CY}. After flagging the error, the 8292 will remain in a loop trying to take control until the current controller stops send-

ing ATN or a new command is written by the microprocessor. If a new command is written, the 8292 will return to the loop after executing it.

TOUT2 Time Out Error 2 occurs when the transmission between the addressed talker and listener has not started for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 45 t_{CY}. This feature is only enabled when the controller is in the CSBS state.

TOUT3 Time Out Error 3 occurs when the handshake signals are stuck and the 8292 is not succeeding in taking control synchronously for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 1800 t_{CY}. The 8292 will continue checking $\overline{ATNI}$ until it becomes true or a new commnand is received. After performing the new command, the 8292 will return to the $\overline{ATNI}$ checking loop.

USER User error occurs when request to assert IFC or REN was received and the 8292 was not the system controller.

Error Mask Register

D_7							D_0
0	0	USER	0	0	$TOUT_3$	$TOUT_2$	$TOUT_1$

The Error Mask Register is used to mask the interrupt from a particular type of error. Each type of error interrupt is enabled by setting the corresponding bit in the Error Mask Register. This register can be read with the RERM command and written with A_0 low.

Command Register

D_7							D_0
1	1	1	OP	C	C	C	C

Commands are performed by the 8292 whenever a byte is written with A_0 high. There are two categories of commands distinguished by the OP bit (bit 4). The first category is the operation command (OP = 1). These commands initiate some action on the interface bus. The second category is the utility command (OP = 0). These commands are used to aid the communication between the processor and the 8292.

OPERATION COMMANDS

Operation commands initiate some action on the GPIB interface bus. It is using these commands that the control functions such as polling, taking and passing control, and system controller functions are performed.

F0—SPCNI—Stop Counter Interrupts

This command disables the internal counter interrupt so that the 8292 will stop interrupting the master on event counter underflows. However, the counter will continue counting and its contents can still be used.

F1—GIDL—Go To Idle

This command is used during the transfer of control procedure while transferring control to another controller. The 8292 will respond to this command only if it is in the active state. $\overline{ATNO}$ will go high, and $\overline{CIC}$ will be high so that this 8292 will no longer be driving the ATN line on the GPIB interface bus. TCI will be set upon completion.

F2—RST—Reset

This command has the same effect as asserting the external reset on the 8292. For details, refer to the reset procedure described later.

F3—RSTI—Reset Interrupts

This command resets any pending interrupts and clears the error flags. The 8292 will not return to any loop it was in (such as from the time out interrupts).

F4—GSEC—Go To Standby, Enable Counting

The function causes $\overline{ATNO}$ to go high and the counter will be enabled. If the 8292 was not the active controller, this command will exit immediately. If the 8292 is the active controller, the counter will be loaded with the value stored in the Event Counter Register, and the internal interrupt will be enabled so that when the counter reaches zero, the SPI interrupt will be generated. SPI will be generated every 256 counts thereafter until the controller exits the standby state or the SPCNI command is written. An initial count of 256 (zero in the Event Counter Register) will be used if the WEVC command is not executed. If the data transmission does not start, a TOUT2 error will be generated.

F5—EXPP—Execute Parallel Poll

This command initiates a parallel poll by asserting EOI when ATN is already active. TCI will be set at the end of the command. The 8291 should be previously configured as a listener. Upon detection of DAV true, the 8291 enters ACDS and latches the parallel poll response (PPR) byte into its data in register. The master will be interrupted by the 8291 BI interrupt when the PPR byte is available. No interrupts except the $\overline{IBFI}$ will be generated by the 8292. The 8292 will respond to this command only when it is the active controller.

F6—GTSB—Go To Standby

If the 8292 is the active controller, $\overline{\text{ATNO}}$ will go high then TCI will be generated. If the data transmission does not start, a TOUT2 error will be generated.

F7—SLOC—Set Local Mode

If the 8292 is the system controller, then REN will be asserted false and TCI will be set true. If it is not the system controller, the User Error bit will be set in the Error Flag Register.

F8—SREM—Set Interface To Remote Control

This command will set REN true and TCI true if this 8292 is the system controller. If not, the User Error bit will be set in the Error Flag Register.

F9—ABORT—Abort All Operation, Clear Interface

This command will cause IFC to be asserted true for at least 100 μsec if this 8292 is the system controller. If it is in CIDS, it will take control over the bus (see the TCNTR command).

FA—TCNTR—Take Control

The transfer of control procedure is coordinated by the master with the 8291 and 8292. When the master receives a TCT message from the 8291, it should issue the TCNTR command to the 8292. The following events occur to take control:

1) The 8292 checks to see if it is in CIDS, and if not, it exits.
2) Then $\overline{\text{ATNI}}$ is checked until it becomes high. If the current controller does not release ATN for the time specified by the Time Out Register, then a TOUT1 error is generated. The 8292 will return to this loop after an error or any command except the RST and RSTI commands.
3) After the current controller releases ATN, the 8292 will assert $\overline{\text{ATNO}}$ and $\overline{\text{CIC}}$ low.
4) Finally, the TCI interrupt is generated to inform the master that it is in control of the bus.

FC—TCASY—Take Control Asynchronously

TCAS transfers the 8292 from CSBS to CACS independent of the handshake lines. If a bus hangup is detected (by an error flag), this command will force the 8292 to take control (asserting ATN) even if the AH function is not in ANRS (Acceptor Not Ready State). This command should be used very carefully since it may cause the loss of a data byte. Normally, control should be taken synchronously. After checking the controller function for being in the CSBS (else it will exit immediately), $\overline{\text{ATNO}}$ will go low, and a TCI interrupt will be generated.

FD—TCSY—Take Control Synchronously

There are two different procedures used to transfer the 8292 from CSBS to CACS depending on the state of the 8291 in the system. If the 8291 is in "continuous AH cycling" mode (Aux. Reg. A0 = A1 = 1), then the following procedures should be followed:

1) The master microprocessor stops the continuous AH cycling mode in the 8291;
2) The master reads the 8291 Interrupt Status 1 Register;
3) If the END bit is set, the master sends the TCSY command to the 8292;
4) If the END bit was not set, the master reads the 8291 Data In Register and then waits for another BI interrupt from the 8291. When it occurs, the master sends the 8292 the TCSY command.

If the 8291 is not in AH cycling mode, then the master just waits for a BI interrupt and then sends the TCSY command. After the TCSY command has been issued, the 8292 checks for $\overline{\text{CSBS}}$. If $\overline{\text{CSBS}}$, then it exits the routine. Otherwise, it then checks the DAV bit in the GPIB status. When DAV becomes false, the 8292 will wait for at least 1.5 μsec. (T10) and then $\overline{\text{ATNO}}$ will go low. If DAV does not go low, a TOUT3 error will be generated. If the 8292 successfully takes control, it sets TCI true.

FE—STCNI—Start Counter Interrupts

This command enables the internal counter interrupt. The counter is enabled by the GSEC command.

UTILITY COMMANDS

All these commands are either Read or Write to registers in the 8292. Note that writing to the Error Mask Register and the Interrupt Mask Register are done directly.

E1—WTOUT—Write To Time Out Register

The byte written to the data bus buffer (with $A_0 = 0$) following this command will determine the time used for the time out function. Since this function is implemented in software, this will not be an accurate time measurement. This feature is enable or disable by the Error Mask Register. No interrupts except for the $\overline{\text{IBFI}}$ will be generated upon completion.

E2—WEVC—Write To Event Counter

The byte written to the data bus buffer (with $A_0 = 0$) following this command will be loaded into the Event Counter Register and the Event Counter Status for byte counting of EOI counting. Only $\overline{IBFI}$ will indicate completion of this command.

E3—REVC—Read Event Counter Status

This command transfers the contents of the Event Counter into the data bus buffer. A TCI is generated when the data is available in the data bus buffer.

E4—RERF—Read Error Flag Register

This command transfers the contents of the Error Flag Register into the data bus buffer. A TCI is generated when the data is available.

E5—RINM—Read Interrupt Mask Register

This command transfers the contents of the Interrupt Mask Register into the data bus buffer. This register is available to the processor so that it does not need to store this information elsewhere. A TCI is generated when the data is available in the data bus buffer.

E6—RCST—Read Controller Status Register

This command transfers the contents of the Controller Status Register into the data bus buffer and a TCI interrupt is generated.

E7—RBST—Read GPIB Bus Status Register

This command transfers the contents of the GPIB Bus Status Register into the data bus buffer, and a TCI interrupt is generated when the data is available.

E9—RTOUT—Read Time Out Status Register

This command transfers the contents of the Time Out Status Register into the data bus buffer, and a TCI interrupt is generated when the data is available.

EA—RERM—Read Error Mask Register

This command transfers the contents of the Error Mask Register to the data bus buffer so that the processor does not need to store this information elsewhere. A TCI interrupt is generated when the data is available.

Interrupt Acknowledge

SYC	ERR	SRQ	EV	1	IFCR	1	1

D_7 D_0

Each named bit in an Interrupt Acknowledge (IACK) corresponds to a flag in the Interrupt Status Register. When the 8292 receives this command, it will clear the SPI and the corresponding bits in the Interrupt Status Register. If not all the bits were cleared, then the SPI will be set true again. If the error flag is not acknowledged by the IACK command, then the Error Flag Register will be transferred to the data bus buffer, and a TCI will be generated.

NOTE:
XXXX1X11 is an undefined operation or utility command, so no conflict exists between the IACK operation and utility commands.

SYSTEM OPERATION

8292 To Master Processor Interface

Communication between the 8292 and the Master Processor can be either interrupt based communication or based upon polling the interrupt status register in predetermined intervals.

Interrupt Based Communication

Four different interrupts are available from the 8292:

OBFI Output Buffer Full Interrupt

$\overline{IBFI}$ Input Buffer Not Full Interrupt

TCI Task Completed Interrupt

SPI Special Interrupt

Each of the interrupts is enabled or disabled by a bit in the interrupt mask register. Since OBFI and $\overline{IBFI}$ are directly connected to the OBF and IBF flags, the master can write a new command to the input data bus buffer as soon as the previous command has been read.

The TCI interrupt is useful when the master is sending commands to the 8292. The pending TCI will be cleared with each new command written to the 8292. Commands sent to the 8292 can be divided into two major groups:

1) Commands that require response back from the 8292 to the master, e.g., reading register.
2) Commands that initiate some action or enable features but do not require response back from the 8292, e.g., enable data bus buffer interrupts.

With the first group, the TCI interrupt will be used to indicate that the required response is ready in the data bus buffer and the master may continue and read it. With the second group, the interrupt will be used to indicate completion of the required task, so that the master may send new commands.

The SPI should be used when immediate information or special events is required (see the Interrupt Status Register).

"Polling Status" Based Communication

When interrupt based communication is not desired, all interrupts can be masked by the interrupt mask register. The communication with the 8292 is based upon sequential poll of the interrupt status register. By testing the OBF and IBF flags, the data bus buffer status is determined while special events are determined by testing the other bits.

Receiving IFC

The IFC pulse defined by the IEEE-488 standard is at least 100 μsec. In this time, all operation on the bus should be aborted. Most important, the current controller (the one that is in charge at that time) should stop sending ATN or EOI. Thus, IFC must externally gate $\overline{\text{CIC}}$ (controller in charge) and $\overline{\text{ATNO}}$ to ensure that this occurs.

Reset and Power Up Procedure

After the 8292 has been reset either by the external reset pin, the device being powered on, or a RST command, the following sequential events will take place:

1) All outputs to the GPIB interface will go high ($\overline{\text{SRQ}}$, $\overline{\text{ATNI}}$, $\overline{\text{IFC}}$, SYC, CLTH, $\overline{\text{ATNO}}$, $\overline{\text{CIC}}$, TCI, SPI, $\overline{\text{EOI}}$, OBFI, $\overline{\text{IBFI}}$, $\overline{\text{DAV}}$, $\overline{\text{REV}}$).
2) The four interrupt outputs (TCI, SPI, OBFI, $\overline{\text{IBFI}}$) and CLTH output will go low.
3) The following registers will be cleared:

   Interrrupt Status

   Interrupt Mask

   Error Flag

   Erorr Mask

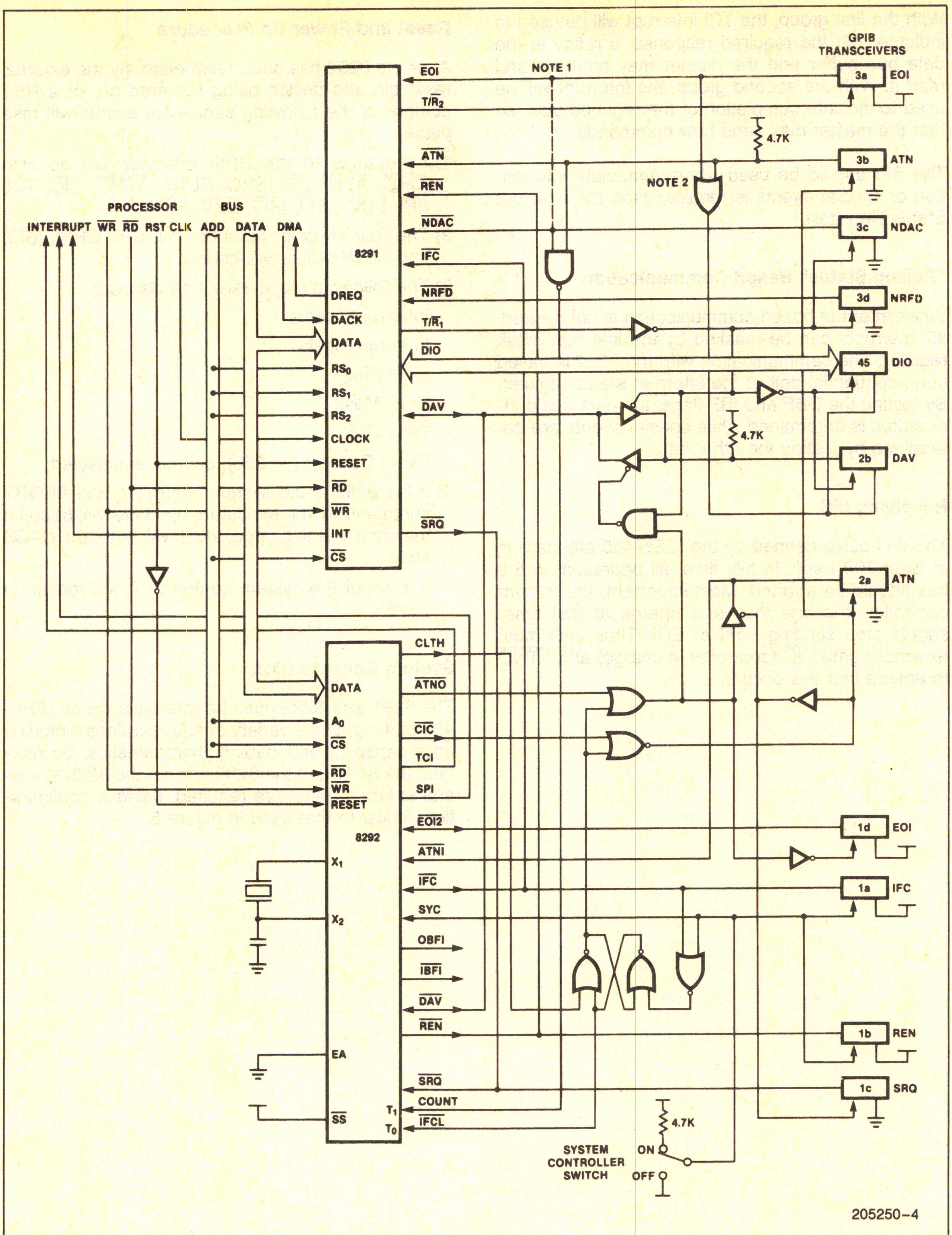
   Time Out

   Event Counter (= 256), counter is disabled.
4) If the 8292 is the system controller, and ABORT command will be executed, the 8292 will become the controller in charge, and it will enter the CACS state.

   If it is not the system controller, it will remain in CIDS.

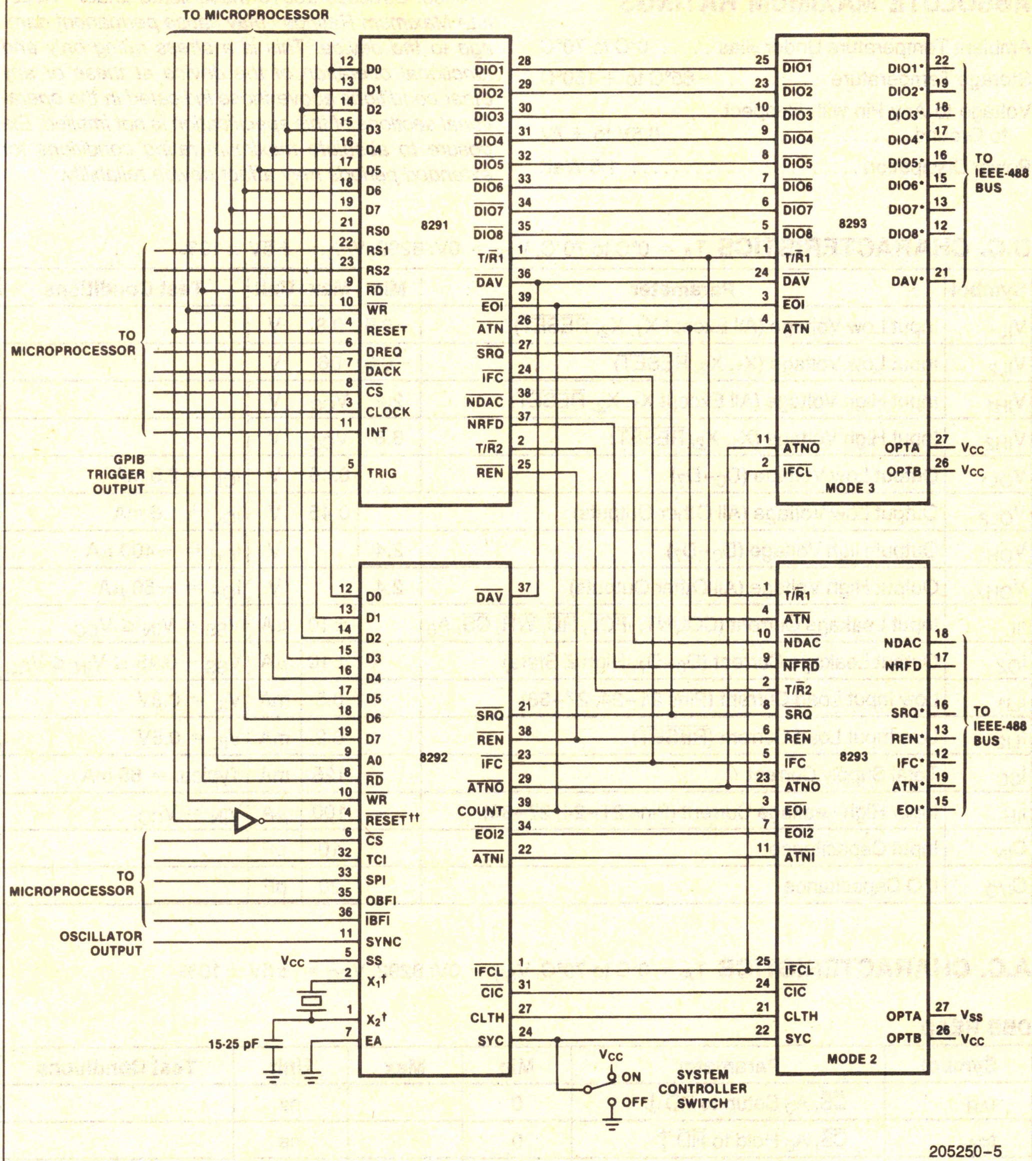
System Configuration

The 8291 and 8292 must be interfaced to an IEEE-488 bus meeting a variety of specifications including drive capability and loading characteristics. To interface the 8291 and the 8292 without the 8293's, several external gates are required, using a configuration similar to that used in Figure 5.



NOTES:
1. Connect to NDAC for byte count or to EOI for block count.
2. Gate ensures open collector operation during parallel poll.

Figure 4. 8291 and 8292 System Configuration



NOTES:

* = GPIB bus transceiver

† = See 8041 data sheet for alternate crystal configurations

†† = Can connect to system reset switch, see 8041A data sheet

Figure 5. 8291, 8292, and 8293 System Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...... 0°C to 70°C
Storage Temperature .... −65°C to +150°C
Voltage to Any Pin with Respect to Ground .... 0.5V to +7V
Power Dissipation .... 1.5 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to 70°C, $V_{SS} = 0V$: 8292, $V_{CC} = \pm 5V \pm 10\%$

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{IL1}	Input Low Voltage (All Except X_1, X_2, $\overline{RESET}$)	−0.5	0.8	V	
V_{IL2}	Input Low Voltage (X_1, X_2, $\overline{RESET}$)	−0.5	06	V	
V_{IH1}	Input High Voltage (All Except X_1, X_2, $\overline{RESET}$)	2.2	V_{CC}	V	
V_{IH2}	Input High Voltage (X_1, X_2, $\overline{RESET}$)	3.8	V_{CC}	V	
V_{OL1}	Output Low Voltage (D_0–D_7)		0.45	V	$I_{OL} = 2.0$ mA
V_{OL2}	Output Low Voltage (All Other Outputs)		0.45	V	$I_{OL} = 1.6$ mA
V_{OH1}	Output High Voltage (D_0–D_7)	2.4		V	$I_{OH} = -400\ \mu A$
V_{OH2}	Output High Voltage (All Other Outputs)	2.4		V	$I_{OH} = -50\ \mu A$
I_{IL}	Input Leakage Current (COUNT, $\overline{IFCL}$, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_0		±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OZ}	Output Leakage Current (D_0–D_7, High Z State)		±10	μA	$V_{SS} + 0.45 \leq V_{IN} \leq V_{CC}$
I_{LI1}	Low Input Load Current (Pins 21–24, 27–38)		0.5	mA	$V_{IL} = 0.8V$
I_{LI2}	Low Input Load Current ($\overline{RESET}$)		0.2	mA	$V_{IL} = 0.8V$
I_{CC}	Total Supply Current		125	mA	Typical = 65 mA
I_{IH}	Input High Leakage Current (Pins 21–24, 27–38)		100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance		10	pF	
$C_{I/O}$	I/O Capacitance		20	pF	

A.C. CHARACTERISTICS $T_A = 0°C$ to 70°C, $V_{SS} = 0V$: 8292, $V_{CC} = \pm 5V \pm 10\%$

DBB READ

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}\downarrow$	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold to $\overline{RD}\uparrow$	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		225	ns	$C_L = 150$ pF
t_{RD}	$\overline{RD}\downarrow$ to Data Out Delay		225	ns	$C_L = 150$ pF
t_{DF}	$\overline{RD}\uparrow$ to Data Float Delay		100	ns	
t_{CY}	Cycle Time	2.5	15	μs	

DBB WRITE

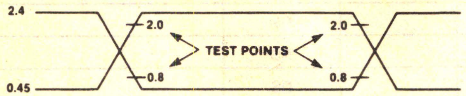
Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	CS, A_0 Setup to WR ↓	0		ns	
t_{WA}	CS, A_0 Hold after WR ↑	0		ns	
t_{WW}	WR Pulse Width	250		ns	
t_{DW}	Data Setup to WR ↑	150		ns	
t_{WD}	Data Hold after WR ↓	0		ns	

COMMAND TIMINGS(1, 3)

Code	Name	Execution Time	$\overline{\text{IBFI}}$ ↑	TCI(2)	SPI	$\overline{\text{ATNO}}$	$\overline{\text{CIC}}$	$\overline{\text{IFC}}$	$\overline{\text{REN}}$	$\overline{\text{EOI}}$	$\overline{\text{DAV}}$	Comments
E1	WTOUT	63	24									
E2	WEVC	63	24									
E3	REVC	71	24	51								
E4	RERF	67	24	47								
E5	RINM	69	24	49								
E6	RCST	97	24	77								
E7	RBST	92	24	72								
E8												
E9	RTOUT	69	24	49								
EA	RERM	69	24	49								
F0	SPCNI	53	24									Count Stops after 39
F1	GIOL	88	24	70		↑61	↑61					
F2	RST	94	24		↓52							Not System Controller
F2	RST	214	24	192	↓52	↓179	↓174	↓101				System Controller
F3	RSTI	61	24									
F4	GSEC	125	24	107		↑98						
F5	EXPP	75	24						↓53 ↑59	↓55 ↑57		
F6	GTSB	118	24	100		↑91						
F7	SLOC	73	24	55				↑46				
F8	SREM	91	24	73				↓64				
F9	ABORT	155	24	133		↓120	↓115	↓42				
FA	TCNTR	108	24	86		↓71	↓68					
FC	TCAS	92	24	67		↓55						
FD	TCSY	115	24	91		↓80						
FE	STCNI	59	24									Starts Count after 43
PIN	RESET	29	—	↓7	↓7							Not System Controller
X	IACK	116	—		↓73 ↑98							If Interrupt Pending

NOTES:
1. All times are multiples of t_{CY} from the 8041A command interrupt.
2. TCI clears after 7 t_{CY} on all commands.
3. ↑ indicates a level transition from low to high, ↓ indicates a high to low transition.

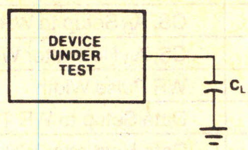
A.C. TESTING INPUT, OUTPUT WAVEFORM



205250-6

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT

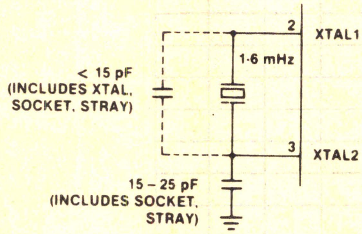


205250-7

C_L Include Jig Capacitance.

CLOCK DRIVER CIRCUITS

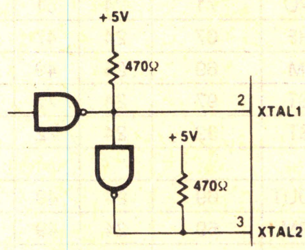
CRYSTAL OSCILLATOR MODE



205250-8

NOTE:
Crystal series resistance should be <75Ω at 6 MHz; <180Ω at 3.6 MHz.

DRIVING FROM EXTERNAL SOURCE

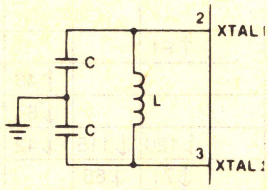


205250-9

NOTE:
Both XTAL1 and XTAL2 should be driven. Resistors to V_{CC} are needed to ensure V_{IH} = 2.8V if TTL circuitry is used.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz



205250-10

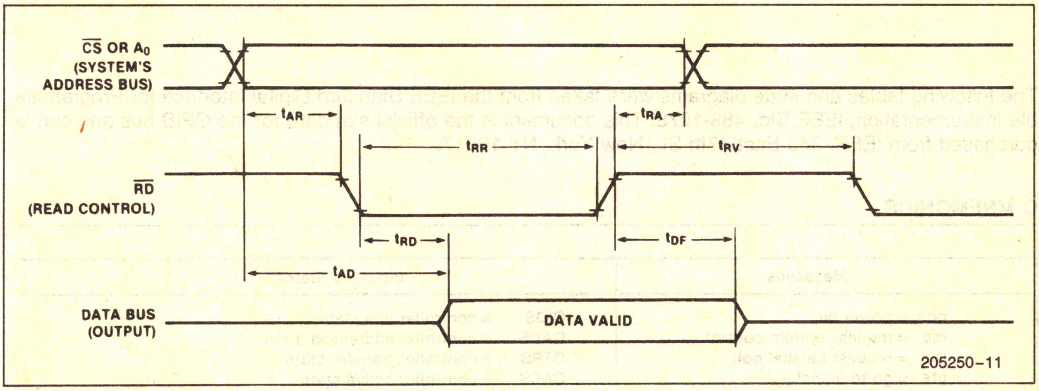
$$f = \frac{1}{2\pi\sqrt{LC'}}$$

$$C' = \frac{C + 3Cpp}{2}$$

NOTES:
1. Cpp ≈ 5–10 pF pin-to-pin capacitance
2. Each C should be approximately 20 pF, including stray capacitance.

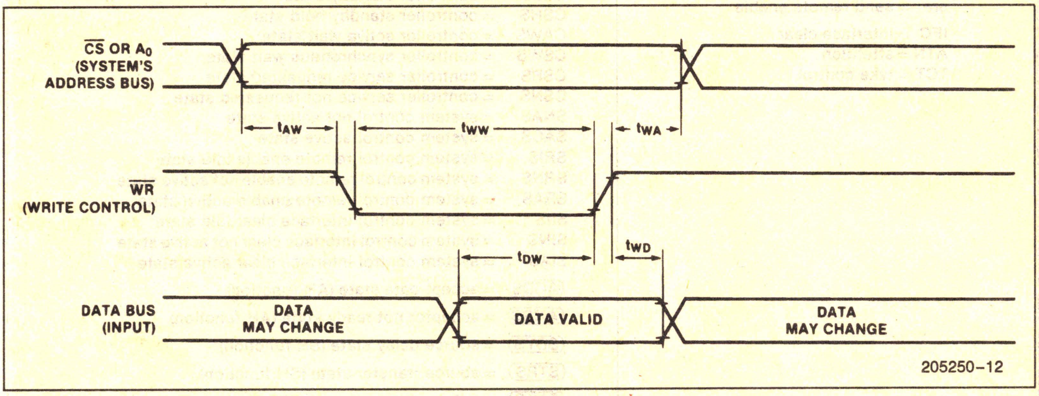
WAVEFORMS

READ OPERATION—DATA BUS BUFFER REGISTER



205250-11

WRITE OPERATION—DATA BUS BUFFER REGISTER



205250-12

APPENDIX A

The following tables and state diagrams were taken from the IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std. 488-1978. This document is the official standard for the GPIB bus and can be purchased from IEEE, 345 East 47th St., New York, NY 10017.

C MNEMONICS

Messages	Interface States
pon = power on	CIDS = controller idle state
rsc = request system control	CADS = controller addressed state
rpp = request parallel poll	CTRS = controller transfer state
gts = go to standby	CACS = controller active state
tca = take control asynchronously	CPWS = controller parallel poll wait state
tcs = take control synchronously	CPPS = controller parallel poll state
sic = send interface clear	CSBS = controller standby state
sre = send remote enable	CSHS = controller standby hold state
IFC = interface clear	CAWS = controller active wait state
ATN = attention	CSWS = controller synchronous wait state
TCT = take control	CSRS = controller service requested state
	CSNS = controller service not requested state
	SNAS = system control not active state
	SACS = system control active state
	SRIS = system control remote enable idle state
	SRNS = system control remote enable not active state
	SRAS = system control remote enable active state
	SIIS = system control interface clear idle state
	SINS = system control interface clear not active state
	SIAS = system control interface clear active state
	ACDS = accept data state (AH function)
	ANRS = acceptor not ready state (AH function)
	SDYS = source delay state (SH function)
	STRS = source transfer state (SH function)
	TADS = talker addressed state (T function)

205250-14

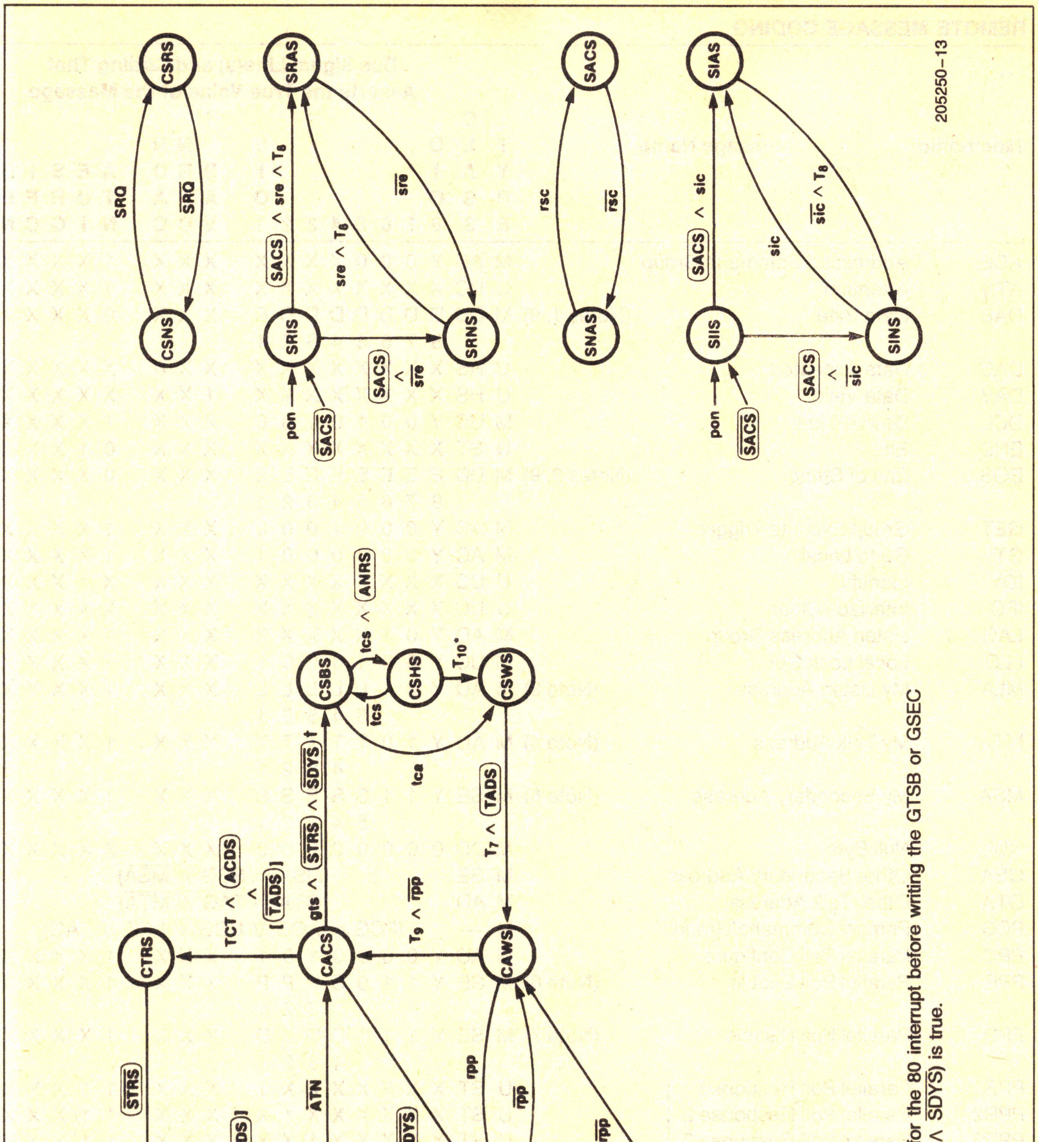


Figure A-1. C State Diagram

REMOTE MESSAGE CODING

Mnemonic	Message Name		TYPE	CLASS	DIO 8	7	6	5	4	3	2	DIO 1	DAV	NRFD	NDAC	ATN	EOI	SRQ	IFC	REN
					Bus Signal Line(s) and Coding That Asserts the True Value of the Message															
ACG	Addressed Command Group		M	AC	Y	0	0	0	X	X	X	X	X	X	X	1	X	X	X	X
ATN	Attention		U	UC	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X
DAB	Data Byte	(Notes 1, 9)	M	DD	D 8	D 7	D 6	D 5	D 4	D 3	D 2	D 1	X	X	X	0	X	X	X	X
DAC	Data Accepted		U	HS	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X
DAV	Data Valid		U	HS	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X
DCL	Device Clear		M	UC	Y	0	0	1	0	1	0	0	X	X	X	1	X	X	X	X
END	End		U	ST	X	X	X	X	X	X	X	X	X	X	X	0	1	X	X	X
EOS	End of String	(Notes 2, 9)	M	DD	E 8	E 7	E 6	E 5	E 4	E 3	E 2	E 1	X	X	X	0	X	X	X	X
GET	Group Execute Trigger		M	AC	Y	0	0	0	1	0	0	0	X	X	X	1	X	X	X	X
GTL	Go to Local		M	AC	Y	0	0	0	0	0	0	1	X	X	X	1	X	X	X	X
IDY	Identify		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X
IFC	Interface Clear		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X
LAG	Listen Address Group		M	AD	Y	0	1	X	X	X	X	X	X	X	X	1	X	X	X	X
LLO	Local Lock Out		M	UC	Y	0	0	1	0	0	0	1	X	X	X	1	X	X	X	X
MLA	My Listen Address	(Note 3)	M	AD	Y	0	1	L 5	L 4	L 3	L 2	L 1	X	X	X	1	X	X	X	X
MTA	My Talk Address	(Note 4)	M	AD	Y	1	0	T	T 4	T 3	T 2	T 1	X	X	X	1	X	X	X	X 5
MSA	My Secondary Address	(Note 5)	M	SE	Y	1	1	S 5	S 4	S 3	S 2	S 1	X	X	X	1	X	X	X	X
NUL	Null Byte		M	DD	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X
OSA	Other Secondary Address		M	SE	(OSA = SCG ∧ $\overline{MSA}$)															
OTA	Other Talk Address		M	AD	(OTA = TAG ∧ $\overline{MTA}$)															
PCG	Primary Command Group		M	—	(PCG = ACG ∨ UCG ∨ LAG ∨ TAG)															
PPC	Parallel Poll Configure		M	AC	Y	0	0	0	0	1	0	1	X	X	X	1	X	X	X	X
PPE	Parallel Poll Enable	(Note 6)	M	SE	Y	1	1	0	S	P 3	P 2	P 1	X	X	X	1	X	X	X	X
PPD	Parallel Poll Disable	(Note 7)	M	SE	Y	1	1	1	D 4	D 3	D 2	D 1	X	X	X	1	X	X	X	X
PPR1	Parallel Poll Response 1	(Note 10)	U	ST	X	X	X	X	X	X	X	1	X	X	X	1	1	X	X	X
PPR2	Parallel Poll Response 2	(Note 10)	U	ST	X	X	X	X	X	X	1	X	X	X	X	1	1	X	X	X
PPR3	Parallel Poll Response 3	(Note 10)	U	ST	X	X	X	X	X	1	X	X	X	X	X	1	1	X	X	X
PPR4	Parallel Poll Response 4	(Note 10)	U	ST	X	X	X	X	1	X	X	X	X	X	X	1	1	X	X	X
PPR5	Parallel Poll Response 5	(Note 10)	U	ST	X	X	X	1	X	X	X	X	X	X	X	1	1	X	X	X
PPR6	Parallel Poll Response 6	(Note 10)	U	ST	X	X	1	X	X	X	X	X	X	X	X	1	1	X	X	X
PPR7	Parallel Poll Response 7	(Note 10)	U	ST	X	1	X	X	X	X	X	X	X	X	X	1	1	X	X	X
PPR8	Parallel Poll Response 8	(Note 10)	U	ST	1	X	X	X	X	X	X	X	X	X	X	1	1	X	X	X
PPU	Parallel Poll Unconfigure		M	UC	Y	0	0	1	0	1	0	1	X	X	X	1	X	X	X	X
REN	Remote Enable		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
RFD	Ready for Data		U	HS	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X	X
RQS	Request Service	(Note 9)	U	ST	X	1	X	X	X	X	X	X	X	X	X	0	X	X	X	X
SCG	Secondary Command Group		M	SE	Y	1	1	X	X	X	X	X	X	X	X	1	X	X	X	X
SDC	Selected Device Clear		M	AC	Y	0	0	0	0	1	0	0	X	X	X	1	X	X	X	X
SPD	Serial Poll Disable		M	UC	Y	0	0	1	1	0	0	1	X	X	X	1	X	X	X	X

REMOTE MESSAGE CODING (Continued)

Mnemonic	Message Name	TYPE	CLASS	DIO 8	7	6	5	4	3	2	DIO 1	DAV	NRFD	NDAC	ATN	EOI	SRQ	IFC	REN
				Bus Signal Line(s) and Coding That Asserts the True Value of the Message															
SPE	Serial Poll Enable	M	UC	Y	0	0	1	1	0	0	0	X	X	X	1	X	X	X	X
SRQ	Service Request	U	ST	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X
STB	Status Byte (Notes 8, 9)	M	ST	S_8	X	S_6	S_5	S_4	S_3	S_2	S_1	X	X	X	0	X	X	X	X
TCT	Take Control	M	AC	Y	0	0	0	1	0	0	1	X	X	X	1	X	X	X	X
TAG	Talk Address Group	M	AD	Y	1	0	X	X	X	X	X	X	X	X	1	X	X	X	X
UCG	Universal Command Group	M	UC	Y	0	0	1	X	X	X	X	X	X	X	1	X	X	X	X
UNL	Unlisten	M	1D	Y	0	1	1	1	1	1	1	X	X	X	1	X	X	X	X
UNT	Untalk (Note 11)	M	1D	Y	1	0	1	1	1	1	1	X	X	X	1	X	X	X	X

The 1/0 coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

1. D1–D8 specify the device dependent data bits.
2. E1–E8 specify the device dependent code used to indicate the EOS message.
3. L1–L5 specify the device dependent bits of the device's listen address.
4. T1–T5 specify the device dependent bits of the device's talk address.
5. S1–S5 specify the device dependent bits of the device's secondary address.
6. S specifies the sense of the PPR.
   Response = $\overline{S \oplus ist}$

P1–P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
0	0	0	PPR1
.	.	.	.
.	.	.	.
.	.	.	.
1	1	1	PPR8

7. D1–D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
8. S1–S6, S8 specify the device dependent status (DIO7 is used for the RQS message.)
9. The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
10. The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
11. This code is provided for system use.

8294A
DATA ENCRYPTION/DECRYPTION UNIT

- **Certified by National Bureau of Standards**
- **400 Byte/Sec Data Conversion Rate**
- **64-Bit Data Encryption Using 56-Bit Key**
- **DMA Interface**
- **3 Interrupt Outputs to Aid in Loading and Unloading Data**
- **7-Bit User Output Port**
- **Single 5V ± 10% Power Supply**
- **Fully Compatible with iAPX-86, 88, MCS-85™, MCS-80™, MCS-51™, and MCS-48™ Processors**
- **Implements Federal Information Processing Data Encryption Standard**
- **Encrypt and Decrypt Modes Available**

The Intel® 8294A Data Encryption Unit (DEU) is a microprocessor peripheral device designed to encrypt and decrypt 64-bit blocks of data using the algorithm specified in the Federal Information Processing Data Encryption Standard. The DEU operates on 64-bit text words using a 56-bit user-specified key to produce 64-bit cipher words. The operation is reversible: if the cipher word is operated upon, the original text word is produced. The algorithm itself is permanently contained in the 8294A; however, the 56-bit key is user-defined and may be changed at any time.

The 56-bit key and 64-bit message data are transferred to and from the 8294A in 8-bit bytes by way of the system data bus. A DMA interface and three interrupt outputs are available to minimize software overhead associated with data transfer. Also, by using the DMA interface two or more DEUs may be operated in parallel to achieve effective system conversion rates which are virtually any multiple of 400 bytes/second. The 8294A also has a 7-bit TTL compatible output port for user-specified functions.

Because the 8294A implements the NBS encryption algorithm it can be used in a variety of Electronic Funds Transfer applications as well as other electronic banking and data handling applications where data must be encrypted.

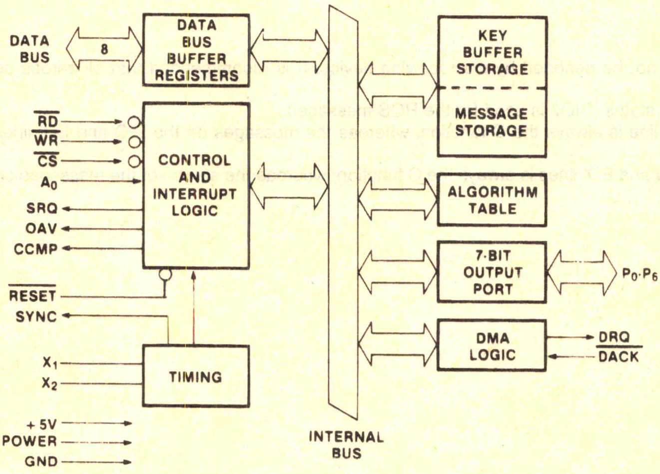
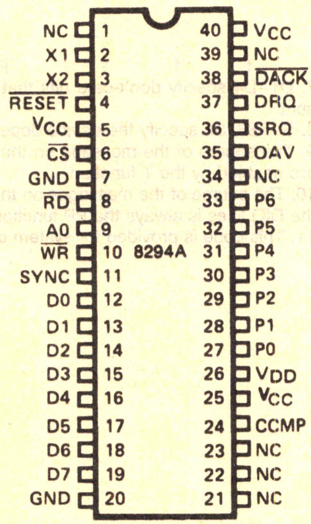


Figure 1. Block Diagram



210465-2

Figure 2. Pin Configuration

November 1986
Order Number: 210465-004

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
NC	1		**NO CONNECTION.**
X1 X2	2 3		**CRYSTAL:** Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency.
$\overline{RESET}$	4	I	**RESET:** A low signal to this pin resets the 8294A.
V_{CC}	5		**POWER:** Tied high.
$\overline{CS}$	6	I	**CHIP SELECT:** A low signal to this pin enables reading and writing to the 8294A.
GND	7		**GROUND:** This pin must be tied to ground.
$\overline{RD}$	8	I	**READ:** An active low read strobe at this pin enables the CPU to read data and status from the internal DEU registers.
A_0	9	I	**ADDRESS:** Address input used by the CPU to select DEU registers during read and write operations.
$\overline{WR}$	10	I	**WRITE:** An active low write strobe at this pin enables the CPU to send data and commands to the DEU.
SYNC	11	O	**SYNC:** High frequency (Clock ÷ 15) output. Can be used as a strobe for external circuitry.
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7	12 13 14 15 16 17 18 19	I/O	**DATA BUS:** Three-state, bi-directional data bus lines used to transfer data between the CPU and the 8294A.
GND	20		**GROUND:** This pin must be tied to ground.
V_{CC}	40		**POWER:** +5V power input: +5V ±10%.
NC	39		**NO CONNECTION.**
$\overline{DACK}$	38	I	**DMA ACKNOWLEDGE:** Input signal from the 8257 DMA Controller acknowledging that the requested DMA cycle has been granted.
DRQ	37	O	**DMA REQUEST:** Output signal to the 8257 DMA Controller requesting a DMA cycle.
SRQ	36	O	**SERVICE REQUEST:** Interrupt to the CPU indicating that the 8294A is awaiting data or commands at the input buffer. SRQ = 1 implies IBF = 0.
OAV	35	O	**OUTPUT AVAILABLE:** Interrupt to the CPU indicating that the 8294A has data or status available in its output buffer, OAV = 1 implies OBF = 1.
NC	34		**NO CONNECTION.**

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
P6 P5 P4 P3 P2 P1 P0	33 32 31 30 29 28 27	O	**OUTPUT PORT:** User output port lines. Output lines available to the user via a CPU command which can asset selected port lines. These lines have nothing to do with the encryption function. At power-on, each line is in a 1 state.
V_{DD}	26		**POWER:** +5V power input. (+5V ±10%) Low power standby pin.
V_{CC}	25		**POWER:** Tied high.
CCMP	24	O	**CONVERSION COMPLETE:** Interrupt to the CPU indicating that the encryption/decryption of an 8-byte block is complete.
NC	23		**NO CONNECTION.**
NC	22		**NO CONNECTION.**
NC	21		**NO CONNECTION.**

FUNCTIONAL DESCRIPTION

OPERATION

The data conversion sequence is as follows:

1) A Set Mode command is given, enabling the desired interrupt outputs.
2) An Enter New Key command is issued, followed by 8 data inputs which are retained by the DEU for encryption/decryption. Each byte must have odd parity.
3) An Encrypt Data or Decrypt Data command sets the DEU in the desired mode.

After this, data conversions are made by writing 8 data bytes and then reading back 8 converted data bytes. Any of the above commands may be issued between data conversions to change the basic operation of the DEU; e.g., a Decrypt Data command could be issued to change the DEU from encrypt mode to decrypt mode without changing either the key or the interrupt outputs enabled.

INTERNAL DEU REGISTERS

Four internal registers are addressable by the master processor: 2 for input, and 2 for output. The following table describes how these registers are accessed.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_0	Register
1	0	0	0	Data Input Buffer
0	1	0	0	Data Output Buffer
1	0	0	1	Command Input Buffer
0	1	0	1	Status Output Buffer
X	X	1	X	Don't Care

The functions of each of these registers are described below.

Data Input Buffer—Data written to this register is interpreted in one of three ways, depending on the preceding command sequence.

1) Part of a key.
2) Data to be encrypted or decrypted.
3) A DMA block count.

Data Output Buffer—Data read from this register is the output of the encryption/decryption operation.

Command Input Buffer—Commands to the DEU are written into this register. (See command summary below.)

Status Output Buffer—DEU status is available in this register at all times. It is used by the processor for poll-driven command and data transfer operations.

STATUS BIT:	7	6	5	4	3	2	1	0
FUNCTION:	X	X	X	KPE	CF	DEC	IBF	OBF

OBF Output Buffer Full; OBF = 1 indicates that output from the encryption/decryption function is available in the Data Output Buffer. It is reset when the data is read.

IBF Input Buffer Full; A write to the Data Input Buffer or to the Command Input Buffer sets IBF = 1. The DEU resets this flag when it has accepted the input byte. Nothing should be written when IBF = 1.

DEC Decrypt; indicates whether the DEU is in an encrypt or a decrypt mode. DEC = 1 implies the decrypt mode. DEC = 0 implies the encrypt mode.

After 8294A has accepted a 'Decrypt Data' or 'Encrypt Data' command, 11 cycles are required to update the DEC bit.

CF Completion Flag; This flag may be used to indicate any or all of three events in the data transfer protocol.

1) It may be used in lieu of a counter in the processor routine to flag the end of an 8-byte transfer.
2) It must be used to indicate the validity of the KPE flag.
3) It may be used in lieu of the CCMP interrupt to indicate the completion of a DMA operation.

KPE Key Parity Error; After a new key has been entered, the DEU uses this flag in conjunction with the CF flag to indicate correct or incorrect parity.

COMMAND SUMMARY

1 — Enter New Key

OP CODE:	0	1	0	0	0	0	0	0
	MSB							LSB

This command is followed by 8 data byte inputs which are retained in the key buffer (RAM) to be used in encrypting and decrypting data. These data bytes must have odd parity represented by the LSB.

2 — Encrypt Data

OP CODE:	0	0	1	1	0	0	0	0
	MSB							LSB

This command puts the 8294A into the encrypt mode.

3 — Decrypt Data

OP CODE:	0	0	1	0	0	0	0	0
	MSB							LSB

This command puts the 8294A into the decrypt mode.

4 — Set Mode

OP CODE:	0	0	0	0	A	B	C	D
	MSB							LSB

where:

A is the OAV (Output Available) interrupt enable
B is the SRQ (Service Request) interrupt enable
C is the DMA (Direct Memory Access) transfer enable
D is the CCMP (Conversion Complete) interrupt enable

This command determines which interrupt outputs will be enabled. A "1" in bits A, B, or D will enable the OAV, SRQ, or CCMP interrupts respectively. A "1" in bit C will allow DMA transfers. When bit C is set the OAV and SRQ interrupts should also be enabled (bits A, B = 1). Following the command in which bit C, the DMA bit, is set, the 8294 will expect one data byte to specify the number of 8-byte blocks to be converted using DMA.

5 — Write to Output Port

OP CODE:	1	P_6	P_5	P_4	P_3	P_2	P_1	P_0
	MSB							LSB

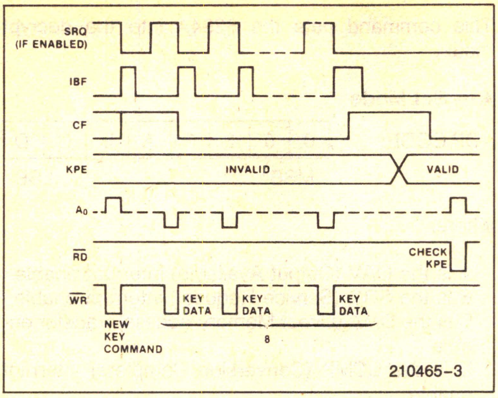
This command causes the 7 least significant bits of the command byte to be latched as output data on the 8294 output port. The initial output is 1111111. Use of this port is independent of the encryption/decryption function.

PROCESSOR/DEU INTERFACE PROTOCOL

ENTERING A NEW KEY

The timing sequence for entering a new key is shown in Figure 3. A flowchart showing the CPU software to accommodate this sequence is given in Figure 4.

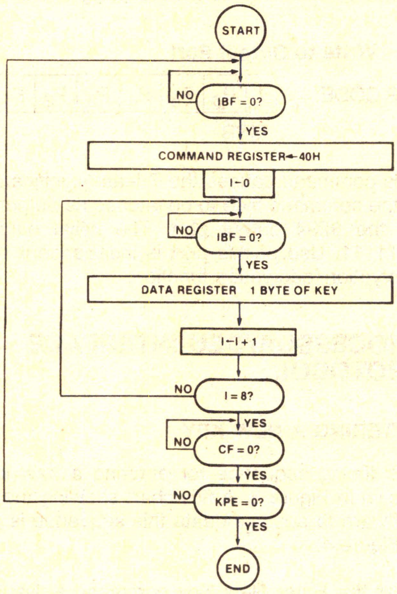
After the Enter New Key command is issued, 8 data bytes representing the new key are written to the data input buffer (most significant byte first). After the eighth byte is entered into the DEU, CF goes true (CF = 1). The CF bit goes false again when KPE is valid. The CPU can then check the KPE flag. If KPE = 1, a parity error has been detected and the DEU has not accepted the key. Each byte is checked for odd parity, where the parity bit is the LSB of each byte.



210465-3

Figure 3. Entering a New Key

Since CF = 1 only for a short period of time after the last byte is accepted, the CPU which polls the CF flag might miss detecting CF = 1 momentarily. Thus, a counter should be used, as in Figure 4, to flag the end of the new key entry. Then CF is used to indicate a valid KPE flag.



210465-4

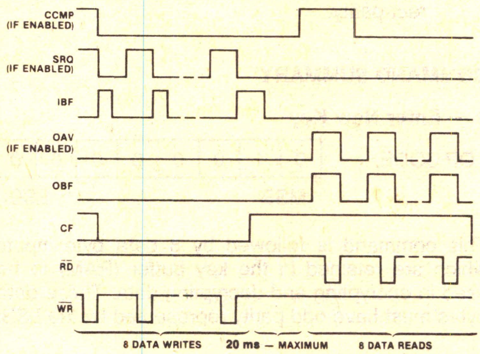
Figure 4. Flowchart for Entering a New Key

ENCRYPTING OR DECRYPTING DATA

Figure 5 shows the timing sequence for encrypting or decrypting data. The CPU writes 8 data bytes to the DEU's data input buffer for encryption/decryption. CF then goes true (CF = 1) to indicate that the DEU has accepted the 8-byte block. Thus, the CPU may test for IBF = 0 and CF = 1 to terminate the input mode, or it may use a software counter. When the encryption/decryption is complete, the CCMP and OAV interrupts are asserted and the OBF flag is set true (OBF = 1). OAV and OBF are set false again after each of the converted data bytes is read back by the CPU. The CCMP interrupt is set false, and remains false, after the first read. After 8 bytes have been read back by the CPU, CF goes false (CF = 0). Thus, the CPU may test for CF = 0 to terminate the read mode. Also, the CCMP interrupt may be used to initiate a service routine which performs the next series of 8 data reads and 8 data writes.

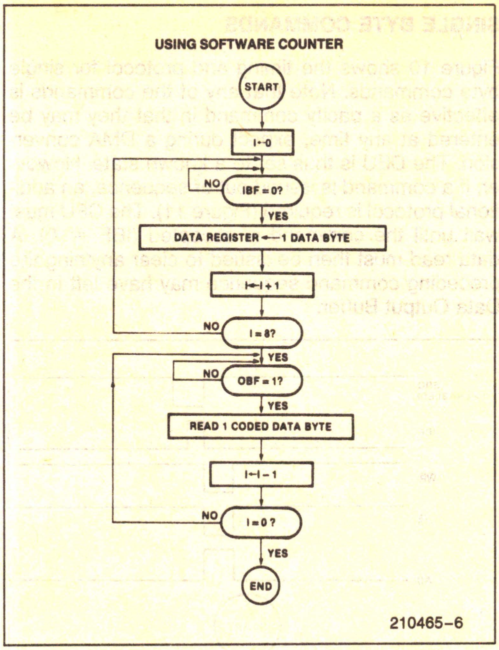
Figure 6 offers two flowcharts outlining the alternative means of implementing the data conversion protocol. Either the CF flag or a software counter may be used to end the read and write modes.

SRQ = 1 implies IBF = 0, OAV = 1 implies OBF = 1. This allows interrupt routines to do data transfers without checking status first. However, the OAV service routine must detect and flag the end of a data conversion.



210465-5

Figure 5. Encrypting/Decrypting Data



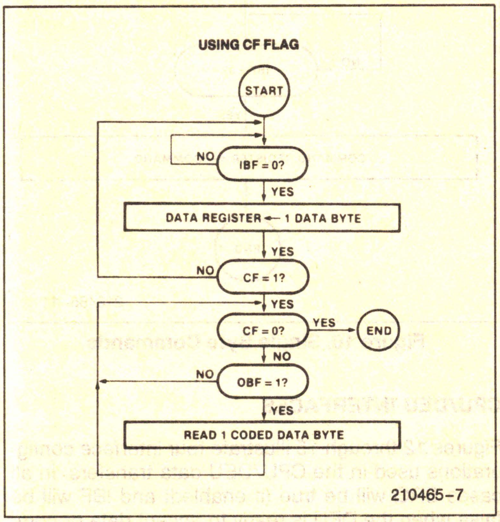


Figure 6. Data Conversion Flowcharts

USING DMA

The timing sequence for data conversions using DMA is shown in Figure 7. This sequence can be better understood when considered in conjunction with the hardware DMA interface in Figure 8. Note

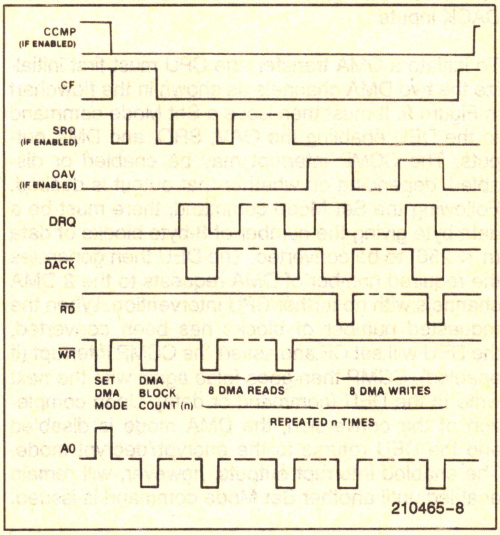


Figure 7. DMA Sequence

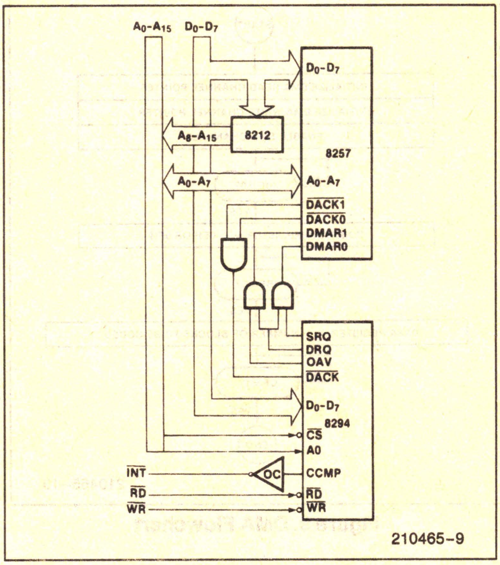


Figure 8. DMA Interface

that the use of the DMA feature requires 3 external AND gates and 2 DMA channels (one for input, one for output). Since the DEU has only one DMA request pin, the SRQ and OAV outputs are used in conjunction with two of the AND gates to create separate DMA request outputs for the 2 DMA channels. The third AND gate combines the two active-low $\overline{DACK}$ inputs.

To initiate a DMA transfer, the CPU must first initialize the two DMA channels as shown in the flowchart in Figure 9. It must then issue a Set Mode command to the DEU enabling the OAV, SRQ, and DMA outputs. The CCMP interrupt may be enabled or disabled, depending on whether that output is desired. Following the Set Mode command, there must be a data byte giving the number of 8-byte blocks of data ($n < 256$) to be converted. The DEU then generates the required number of DMA requests to the 2 DMA channels with no further CPU intervention. When the requested number of blocks has been converted, the DEU will set CF and assert the CCMP interrupt (if enabled). CCMP then goes false again with the next write to the DEU (command or data). Upon completion of the conversion, the DMA mode is disabled and the DEU returns to the encrypt/decrypt mode. The enabled interrupt outputs, however, will remain enabled until another Set Mode command is issued.

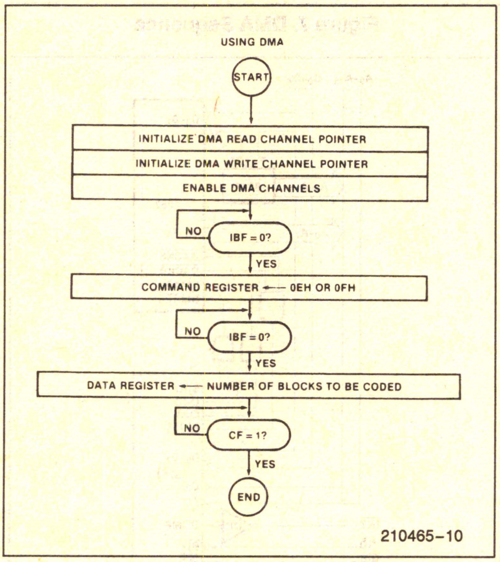


Figure 9. DMA Flowchart

SINGLE BYTE COMMANDS

Figure 10 shows the timing and protocol for single byte commands. Note that any of the commands is effective as a pacify command in that they may be entered at any time, expect during a DMA conversion. The DEU is thus set to a known state. However, if a command is issued out of sequence, an additional protocol is required (Figure 11). The CPU must wait until the command is accepted (IBF = 0). A data read must then be issued to clear anything the preceding command sequence may have left in the Data Output Buffer.

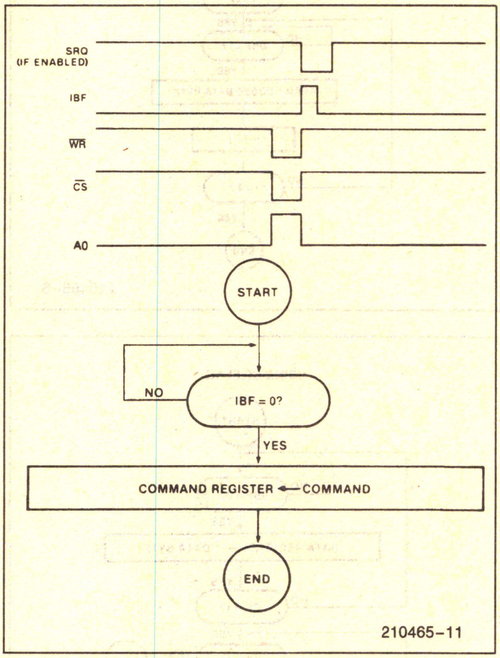


Figure 10. Single Byte Commands

CPU/DEU INTERFACES

Figures 12 through 15 illustrate four interface configurations used in the CPU/DEU data transfers. In all cases SRQ will be true (if enabled) and IBF will be false when the DEU is ready to accept data or commands.

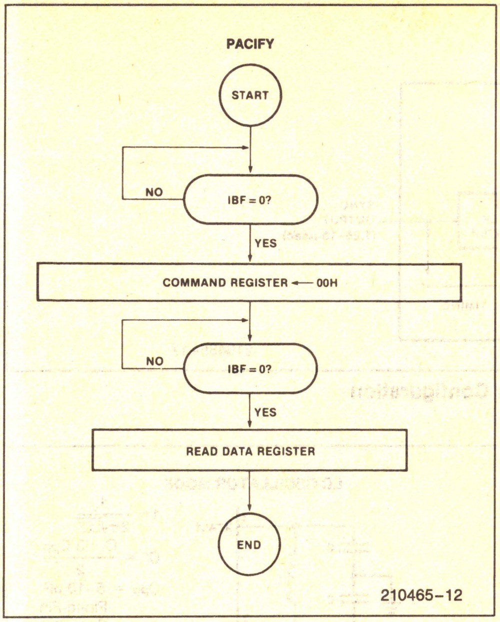


Figure 11. Pacify Protocol

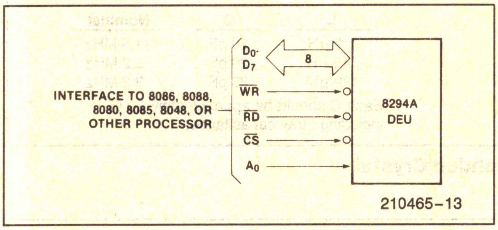


Figure 12. Polling Interface

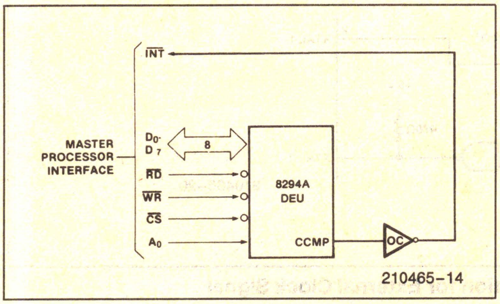


Figure 13. Single Interrupt Interface

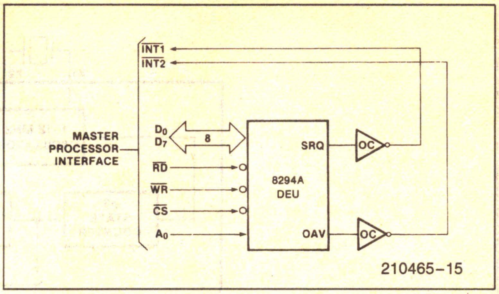
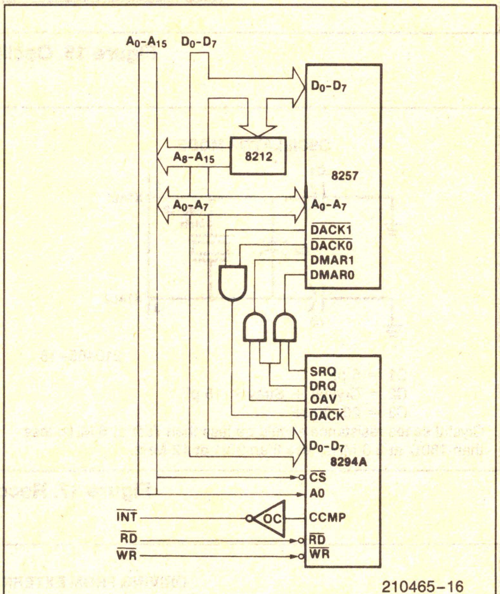


Figure 14. Dual Interrupt Interface



DMAR0 is for memory to DEU Data Transfer
DMAR1 is for DEU to memory Data Transfer
Use of CCMP is optional

Figure 15. DMA Interface

OSCILLATING AND TIMING CIRCUITS

The 8294A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 16.

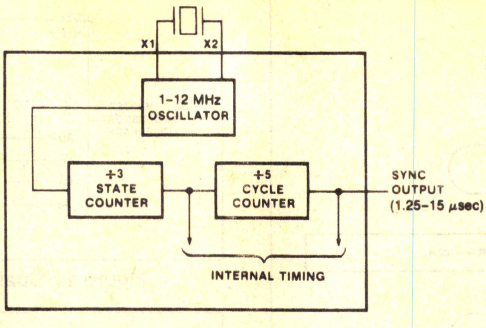
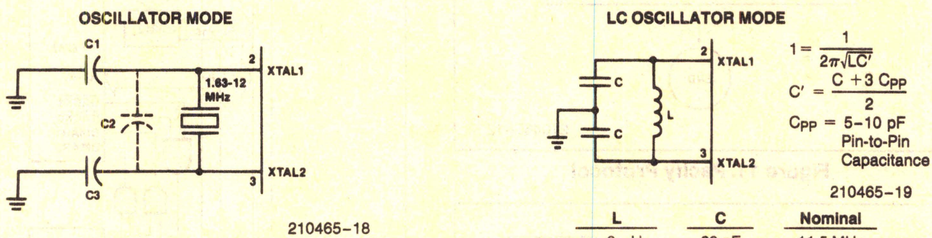


Figure 16. Oscillator Configuration



C1 = 5 pF
C2 = Crystal + Stray < 15 pF
C3 = 20-30 pF
Crystal series resistance should be less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz; less than 30Ω at 12 MHz.

$$1 = \frac{1}{2\pi\sqrt{LC'}}$$

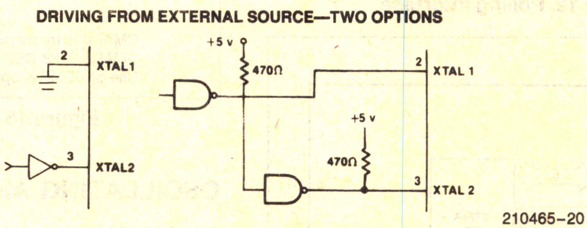
$$C' = \frac{C + 3\,C_{PP}}{2}$$

C_{PP} = 5-10 pF Pin-to-Pin Capacitance

L	C	Nominal
9 μH	20 pF	11.5 MHz
45 μH	20 pF	5.2 MHz
120 μM	20 pF	3.2 MHz

Each C should be approximately 20 pF including stray capacitance.

Figure 17. Recommended Crystal



For the 8294A XTAL2 must be high 35-65% of the period.
Rise and fall times must not exceed 10 ns.
Resistor to V_{CC} is needed to ensure V_{IH} = 3.0V if TTL circuitry is used.

Figure 18. Recommended Connection for External Clock Signal

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to +70°C
Storage Temperature . . . . −65°C to +150°C
Voltage on Any Pin With Respect to Ground . . . . −0.5V to +7V
Power Dissipation . . . . 1.5 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. AND OPERATING CHARACTERISTICS

$T_A = 0°C$ to $+70°C$, $V_{CC} = +5V \pm 10\%$, $V_{SS} = 0V$

Symbol	Parameter	Limits Min	Limits Typ	Limits Max	Unit	Test Conditions
V_{IL}	Input Low Voltage (All Except X_1, X_2, RESET)	−0.5		0.8	V	
V_{IL1}	Input Low Voltage (X_1, X_2, RESET	−0.5		0.6	V	
V_{IH}	Input High Voltage (All Except X_1, RESET)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X_1, RESET	3.5		V_{CC}	V	
V_{IH2}	Input High Voltage (X_2)	2.2		V_{CC}	V	
V_{OL}	Output Low Voltage (D_0–D_7)			0.45	V	$I_{OL} = 2.0$ mA
V_{OL1}	Output Low Voltage (All Other Outputs)			0.45	V	$I_{OL} = 1.6$ mA
V_{OH}	Output High Voltage (D_0–D_7)	2.4			V	$I_{OH} = -400\ \mu A$
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			V	$I_{OH} = -50\ \mu A$
I_{IL}	Input Leakage Current (RD, WR, CS, A_0)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Output Leakage Current (D_0–D_7, High Z State)			±10	μA	$V_{SS} + 0.45 \leq V_{OUT} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		5	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		60	135	mA	
I_{LI}	Low Input Load Current (Pins 24, 27–38)			0.3	mA	$V_{IL} = 0.8V$
I_{LI1}	Low Input Load Current (RESET)			0.2	mA	$V_{IL} = 0.8V$
I_{IH}	Input High Leakage Current (Pins 24, 27–38)			100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance			10	pF	
$C_{I/O}$	I/O Capacitance			20	pF	

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$, $V_{CC} = V_{DD} = +5V \pm 10\%$, $V_{SS} = 0V$

DBB READ

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$ ↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$ ↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	160		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		130	ns	$C_L = 100$ pF
t_{RD}	$\overline{RD}$ ↓ to Data Out Delay		130	ns	$C_L = 100$ pF
t_{DF}	$\overline{RD}$ ↑ to Data Float Delay		85	ns	
t_{CY}	Cycle Time	1.25	15	μs	1–12 MHz Crystal

DBB WRITE

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	$\overline{CS}$, A_0 Setup to $\overline{WR}$ ↓	0		ns	
t_{WA}	$\overline{CS}$, A_0 Hold After $\overline{WR}$ ↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	160		ns	
t_{DW}	Data Setup to $\overline{WR}$ ↑	130		ns	
t_{WD}	Data Hold to $\overline{WR}$ ↑	0		ns	

DMA AND INTERRUPT TIMING

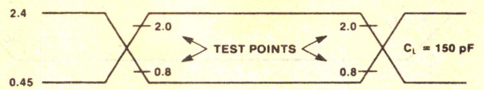
Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{ACC}	$\overline{DACK}$ Setup to Control	0		ns	
t_{CAC}	$\overline{DACK}$ Hold After Control	0		ns	
t_{ACD}	$\overline{DACK}$ to Data Valid		130	ns	$C_L = 100$ pF
t_{CRQ}	Control L.E. to DRQ T.E.		110	ns	
t_{CI}	Control T.E. to Interrupt T.E.		400	ns	

CLOCK

Symbol	Parameter	Min	Max	Units
t_{CY}	Cycle Time	1.25	9.20	μs(1)
t_{CYC}	Clock Period	83.3	613	ns
t_{PWH}	Clock High Time	38		ns
t_{PWL}	Clock Low Time	38		ns
t_R	Clock Rise Time		10	ns
t_F	Clock Fall Time		10	ns

NOTE:
1. $t_{CY} = 15/f(XTAL)$

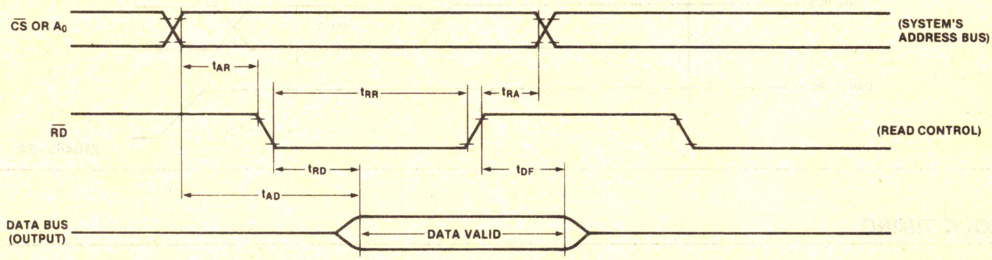
A.C. TESTING INPUT, OUTPUT WAVEFORM



210465-21

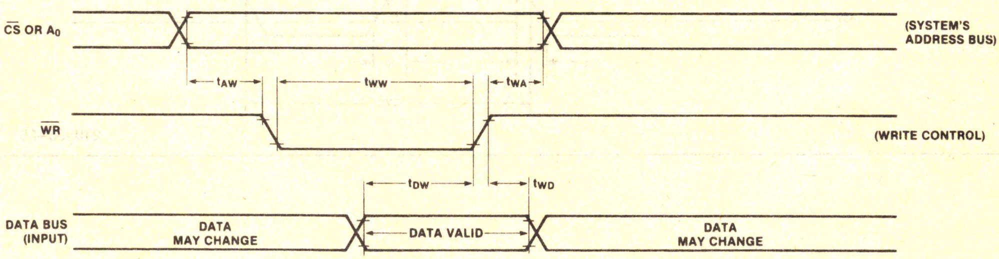
WAVEFORMS

READ OPERATION—OUTPUT BUFFER REGISTER



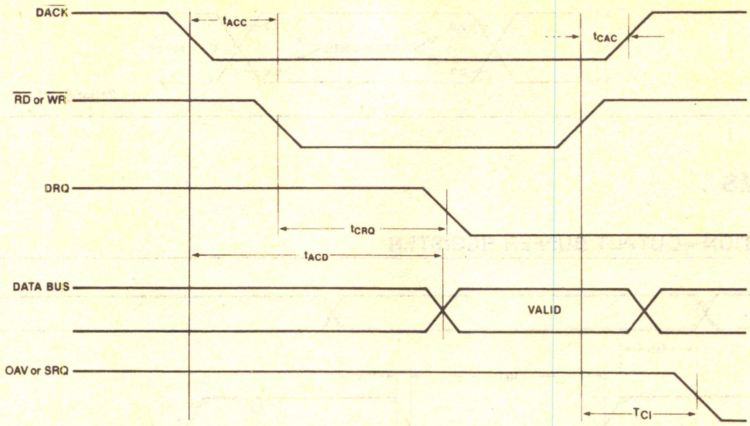
210465-22

WRITE OPERATION—INPUT BUFFER REGISTER



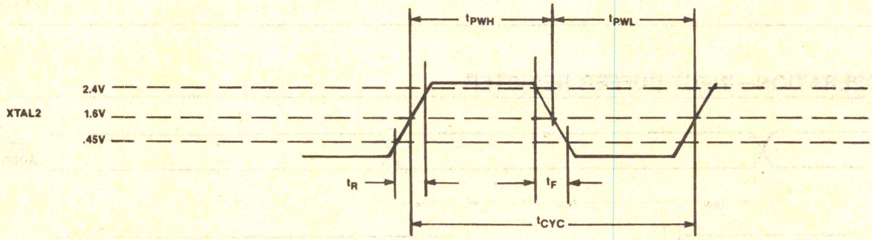
210465-23

DMA AND INTERRUPT TIMING



210465-24

CLOCK TIMING



210465-25

APPLICATION NOTE

AP-166

November 1986

Using the 8291A GPIB Talker/Listener

Order Number: 230832-001

INTRODUCTION

This application note explains the Intel 8291A GPIB (General Purpose Interface Bus) Talker/Listener as a component, and shows its use in GPIB interface design tasks.

The first section of this note presents an overview of IEEE 488 (GPIB). The second section introduces the Intel GPIB component family. A detailed explanation of the 8291A follows. Finally, some application examples using the component family are presented.

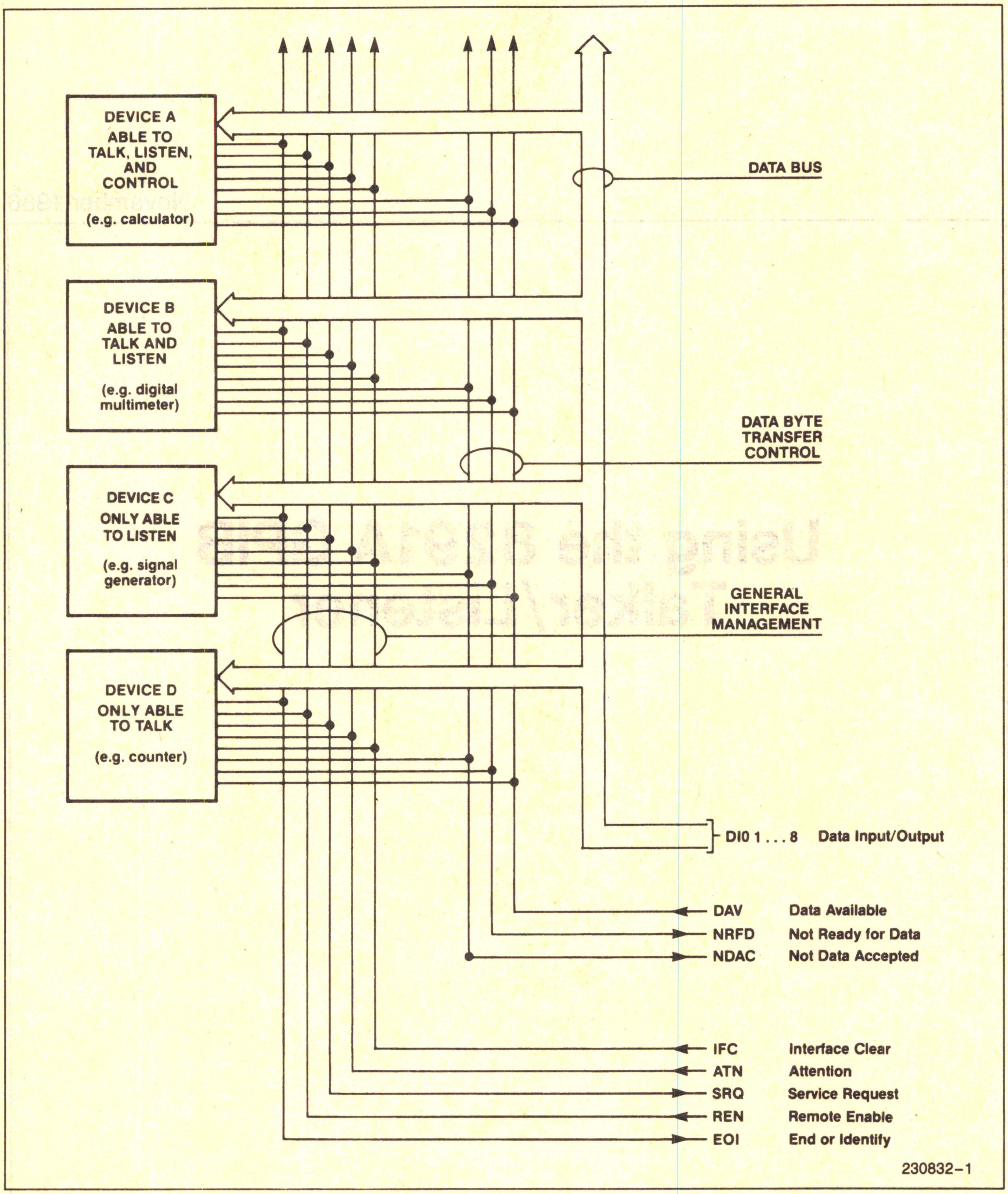


Figure 1. Interface Capabilities and Bus Structure

OVERVIEW OF IEEE 488/GPIB

The GPIB is a parallel interface bus with an asynchronous interlocking data exchange handshake mechanism. It is designed to provide a common communication interface among devices over a maximum distance of 20 meters at a maximum speed of 1 Mbps. Up to 15 devices may be connected together. The asynchronous interlocking handshake dispenses with a common synchronization clock, and allows intercommunication among devices capable of running at different speeds. During any transaction, the data transfer occurs at the speed of the slowest device involved.

The GPIB finds use in a diversity of applications requiring communication among digital devices over short distances. Common examples are: programmable instrumentation systems, computer to peripherals, etc.

The interface is completely defined in the IEEE STD.-488-1978.

A typical implementation consists of logical devices which talk (talker), listen (listeners), and control GPIB activity (controllers).

Interface Functions

The interface between any device and the bus may have a combination of several different capabilities (called 'functions'). Among a total of ten functions defined, the Talker, Listener, Source Handshake, Acceptor Handshake and Controller are the more common examples. The Talker function allows a device to transmit data. The Listener function allows reception. The Source and Acceptor Handshakes, synchronized with the Talker and Listener functions respectively, exchange the handshake signals that coordinate data transfer. The Controller function allows a device to activate the interface functions of the various devices through commands. Other interface functions are: Service request, Remote local, Parallel poll, Device clear and Device trigger. Each interface may not contain all these functions. Further, most of these functions may be implemented to various levels (called 'subsets') of capability. Thus, the overall capability of an interface may be tailored to the needs of the communicating device.

Electrical Signal Lines

As shown in Figure 1, the GPIB is composed of eight data lines (D08–D01), five interface management lines (IFC, ATN, SRQ, REN, EOI), and three transfer control lines (DAV, NRFD, NDAC).

The eight data lines are used to transfer data and commands from one device to another with the help of the management and control lines. Each of the five interface management lines has a specific function.

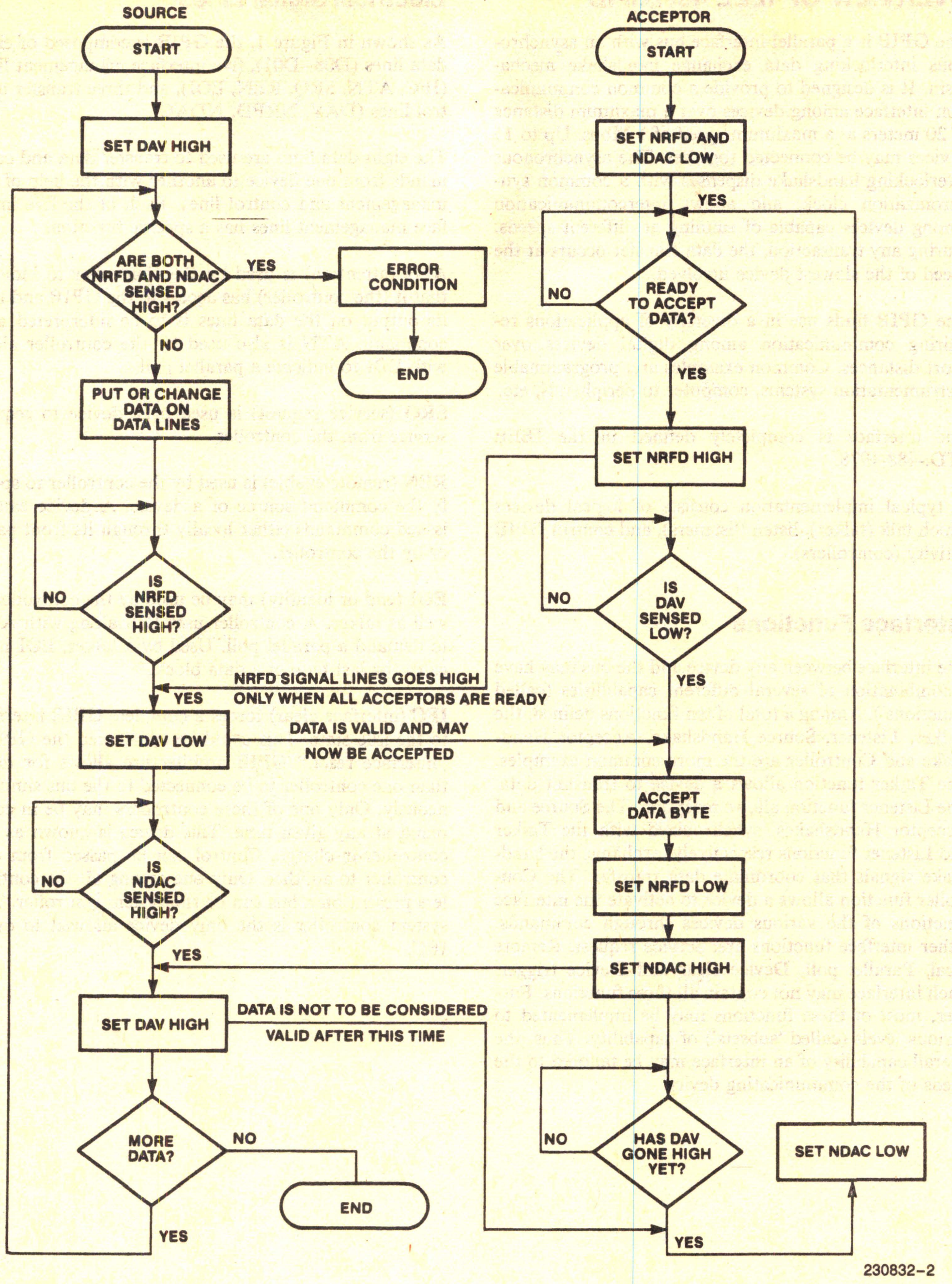
ATN (attention) is used by the Controller to indicate that it (the controller) has access to the GPIB and that its output on the data lines is to be interpreted as a command. ATN is also used by the controller along with EOI to indicate a parallel poll.

SRQ (service request) is used by a device to request service from the controller.

REN (remote enable) is used by the controller to specify the command source of a device. A device can be issued commands either locally through its front panel or by the controller.

EOI (end or identify) may be used by the controller as well as talker. A controller uses EOI along with ATN to demand a parallel poll. Used by a talker, EOI indicates the last byte of a data block.

IFC (interface clear) forces a complete GPIB interface to the idle state. This could be considered the GPIB's "interface reset." GPIB architecture allows for more than one controller to be connected to the bus simultaneously. Only one of these controllers may be in command at any given time. This device is known as the controller-in-charge. Control can be passed from one controller to another. Only one among all the controllers present on a bus can be the system controller. The system controller is the only device allowed to drive IFC.



NOTE:
Flow diagram outlines sequence of events during transfer of data byte. More than one listener at a time can accept data because of logical connection of NRFD and NDAC lines.

Figure 2. Handshake Flowchart

Transfer Control Lines

The transfer control lines conduct the asynchronous interlocking three-wire handshake.

DAV (data valid) is driven by a talker and indicates that valid data is on the bus.

NRFD (note ready for data) is driven by the listeners and indicates that not all listeners are ready for more data.

NDAC (not data accepted) is used by the listeners to indicate that not all listeners have read the GPIB data lines yet.

The asynchronous 3-wire handshake flowchart is shown in Figure 2. This is a concept fundamental to the asynchronous nature of the GPIB and is reviewed in the following paragraphs.

Assume that a talker is ready to start a data transfer. At the beginning of the handshake, NRFD is false indicating that the listener(s) is ready for data. NDAC is true indicating that the listener(s) has not accepted the data, since no data has been sent yet. The talker places data on the data lines, waits for the required settling time, and then indicates valid data by driving DAV true. All active listeners drive NRFD true indicating that they are not ready for more data. They then read the data and drive NDAC false to indicate acceptance. The talker responds by deasserting DAV and readies itself to transfer the next byte. The listeners respond to DAV false by driving NDAC true. The talker can now drive the data lines with a new data byte and wait for NRFD to be false to start the next handshake cycle.

Bus Commands

When ATN and DAV are true data patterns which have been placed by the controller on the GPIB, they are interpreted as commands by the other devices on the interface. The GPIB standard contains a repertory of commands such as MTA (My Talk Address), MSA (My Secondary Address), SPE (Serial Poll Enable), etc. All other patterns in conjunction with ATN and DAV are classified as undefined commands and their meaning is user-dependent.

Addressing Techniques

To allow the controller to issue commands selectively to specific devices, three types of addressing exist on the GPIB: talk only/listen only (ton/lon), primary, and secondary.

Ton/lon is a method where the ability of the GPIB interface to talk or listen is determined by the device and not by the GPIB controller. With this method, fixed poles can be easily designated in simple systems where reassignment is not necessary. This is appropriate and convenient for certain applications. For example, a logic analyzer might by interfaced via the GPIB to a line printer in order to document some type of failure. In this case, the line printer simply listens to the logic analyzer, which is a talker.

The controller addresses devices through three commands, MTA (my talk address), MLA (my listen address), and MSA (my secondary address). The device address is imbedded in the command bit pattern. The device whose address matches the imbedded pattern is enabled. Some devices may have the same logical talk and listen addresses. This is allowable since the talker and listener are separate functions. However, two of the same functions cannot have the same address.

In primary addressing, a device is enabled to talk (listen) by receiving the MTA (MLA) message.

Secondary addressing extends the address field from 5 to 10 bits by allowing an additional byte. This additional byte is passed via the MSA message. Secondary addressing can also be used to logically divide devices into various subgroups. The MSA message applies only to the device(s) whose primary address immediately precede it.

INTEL'S® GPIB COMPONENTS

The logic designer implementing a GPIB interface has, in the past, been faced with a difficult and complex discrete logic design. Advances in LSI technology have produced sophisticated microprocessor and peripheral devices which combine to reduce this once complex interface task to a system consisting of a small set of integrated circuits and some software drivers. A microprocessor hardware/software solution and a high-level language source code provide an additional benefit in end-product maintenance. Product changes are a simple matter of revising the product software. Field changes are as easy as exchanging EPROMS.

Intel has provided an LSI solution to GPIB interfacing with a talker/listener device (8291A), a controller device (8292), and a transceiver (8293). An interface with all capabilities except for the controller function can be built with an 8291A and a pair of 8293's. The addition of the 8292 produces a complex interface. Since most devices in a GPIB system will not have the controller function capability, this modular approach provides the least cost to the majority of interface designs.

Overview of the 8291A GPIB Talker/Listener

The Intel 8291A GPIB Talker/Listener operates over a clock range of 1 to 8 MHz and is compatible with the MCS-85, iAPX-86, and 8051 families of microprocessors.

A detailed description of the 8291A is given in the data sheet.

The 8291A implements the following functions: Source Handshake (SH), Acceptor Handshake (AH), Talker Extended (TE), Service Request (SRQ), Listener Extended (LE), Remote/Local (RL), Parallel Poll (PP2), Device Clear (DC), and Device Trigger (DT).

Current states of the 8291A can be determined by examining the device's status read registers. In addition, the 8291A contains 8 write registers. These registers are shown in Figure 3. The three register select pins RS3–RS0 are used to select the desired register.

The data-in register moves data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. When the 8291A is addressed to talk, it uses the data-out register to move data onto the GPIB. The serial poll mode and status registers are used to request service and program the serial poll status byte.

A detailed description of each of the registers, along with state diagrams can be found in the 8291A data sheet.

Read Registers									RS2	RS1	RS0	Write Registers								
DATA IN	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	0	0	0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0	DATA OUT
INTERRUPT STATUS 1	CPT	APT	GET	END	DEC	ERR	BO	BI	0	0	1	CPT	APT	GET	END	DEC	ERR	BO	BI	INTERRUPT ENABLE 1
INTERRUPT STATUS 2	INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC	0	1	0	0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC	INTERRUPT ENABLE 2
SERIAL POLL STATUS 2	S8	SRQS	S6	S5	S4	S3	S2	S1	0	1	1	S8	RSV	S6	S5	S4	S3	S2	S1	SERIAL POLL MODE
ADDRESS STATUS	ton	Lon	EOI	LPAS	TPAS	LA	TA	MJMN	1	0	0	TO	LO	0	0	0	0	ADM1	ADM0	ADDRESS MODE
COMMAND PASS THROUGH	CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	1	0	1	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0	AUX MODE
ADDRESS 0	INT	DTO	DLO	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0	1	1	0	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1	ADDRESS 0/1
ADDRESS 1	X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1	1	1	1	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0	EOS

Figure 3. 8291A Registers

Address Mode

The address mode and status registers are used to program the addressing modes and track addressing states. The auxiliary mode register is used to select a variety of functions. The command pass through register is used for undefined commands and extended addresses. The address 0/1 register is used to program the addresses to which the 8291A will respond. The address 0 and address 1 registers allow reading of these programmed addresses plus trading of the interrupt bit. The EOS register is used to program the end of sequence character.

Detailed descriptions of the addressing modes available with the 8291A are described in the 8291A data sheet. Examples of how to program these modes are shown below.

1. MODE: Talker has single address of 01H
   Listener has single address of 02H

CPU Writes to:	Pattern	Comment
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0010 0001	Major is Talking. Address = 01H
Address 0/1 Register	1100 0010	Minor is Listener. Address = 02H

2. MODE: Talker has single address of 01H
   Listener has single address of 02H

CPU Writes to:	Pattern	Comment
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0100 0010	Major is Listener. Address = 02H
Address 0/1 Register	1010 0001	Minor is Talking. Address = 01H

Note that in both of the above examples, the listener will respond to a MLA message with five least significant bits equal to 02H and the talker to a 01H.

3. MODE: Talker and listener both share a single address of 03H

CPU Writes to:	Pattern	Comment
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0000 0011	Talker and Listener Address = 03
Address 0/1 Register	1110 0000	Minor Address is disabled

4. MODE: Talker and listener have a primary address of 04H and a secondary address of 05H

CPU Writes to:	Pattern	Comment
Address Mode Register	0000 0010	Select Mode 2 Addressing
Address 0/1 Register	0000 0100	Primary Address = 04H
Address 0/1 Register	1000 0101	Minor Address is disabled

5. MODE: Talker has a primary address of 06H. Listener has a primary address of 07H

CPU Writes to:	Pattern	Comment
Address Mode Register	0000 0011	Select Mode 3
Address 0/1 Register	0010 0110	Talker Address = 06
Address 0/1 Register	1100 0111	Listener Primary = 07

The CPU will verify the secondary addresses which could be the same or different.

APPLICATION OF THE 8291A

This phase of the application note will examine programming of the 8291A, corresponding bus commands and responses, CPU interruption, etc. for a variety of GPIB activities. This should provide the reader with a clear understanding of the role of the 8291A performs in a GPIB system. The talker function, listener function, remote message handling, and remote/local operations including local lockout, are discussed.

Talker Functions

TALK-ONLY (ton). In talk only mode the 8291A will not respond to the MTA message from a controller. Generally, ton is used in an environment which does not have a controller. Ton is also employed in an interface that includes the controller function.

When the 8291A is used with the 8292, the sequence of events for initialization are as follows:

1) The Interrupt/Enable registers are programmed.
2) Ton is selected.
3) Settling time is selected.
4) EOS character is loaded.
5) "Pon" local message is sent.
6) CPU waits for Byte Out (BO) and sends a byte to the data out register.

Addressed Talker (via MTA Message)

The GPIB controller will direct the 8291A to talk by sending a My Talk Address (MTA) message containing the 8291A's talk address. The sequence of events is as follows:

1) The interrupt enable and serial poll mode registers are programmed.
2) Mode 1 is selected.
3) Settling time is selected.
4) Talker and listener addresses are programmed.
5) Power on (pon) local message is sent.
6) CPU waits for an interrupt. When the controller has sent the MTA message for the 8291A an interrupt will be generated if enabled and the ADSC bit will be set.
7) CPU reads the Address Status register to determine if the 8291A has been addressed to talk (TA = 1).
8) CPU waits for an interrupt from either BO or ADSC
9) When BO is set, the CPU writes the data byte to the data out register.
10) CPU continues to poll the status registers.
11) When unaddressed ADSC, will be set and TA reset.

LISTENER FUNCTIONS

LISTEN-ONLY (lon). In listen-only mode the 8291 will not respond to the My Listen Address (MLA) message from the controller. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) Lon is selected.
3) EOS character is programmed.
4) "Pon" local message is sent.
5) CPU waits for BI and reads the byte from the data-in register.

Note that enabling both ton and lon can create an internal loopback as long as another listener exists.

Addressed Listening (via the MLA Message)

The GPIB controller will direct the 8291A to listen by sending a MLA message containing the 8291A's listen address. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) The serial poll mode register is loaded as desired.
3) Talker and listener addresses are loaded.
4) "Pon" local message is sent.
5) The CPU waits for an interrupt. When the controller has sent the MLA message for the 8291A, the ADSC bit will be set.
6) The CPU reads the Address Status Register to determine if the 8291A has been addressed to listen (LA = 1).
7) CPU waits for an interrupt for BI or ADSC.
8) When BI is set, the CPU reads the data byte from the data-in register.
9) The CPU continues to poll the status registers.
10) When unaddressed, ADSC will be set and LA reset.

Remote/Local and Lockout

Remote and local refer to the source of control of a device connected to the GPIB. Remote refers to control from the GPIB controller-in-charge. Local refers to control from the device's own system. Reference should be made to the RL state diagram in the 2891A data sheet.

Upon "pon" the 8291A is in the local state. In this state the REM bit in Interrupt Status 1 Register is reset. When the GPIB controller takes control of the bus it will drive the REN (remote enable) line true. This will cause the REM bit and REMC (remote/local change) bit to be set. The distinction between remote and local modes is necessary in that some types of devices will have local controls which have functions which are also controlled by remote messages.

In the local state the device is allowed to store, but not respond to, remote messages which control functions which are also controlled by local messages. A device which has been addressed to listen will exit the local state and go to the remote state if the REN message is true and the local rtl (return to local) message is false. The state of the "rtl" local message is ignored and the device is "locked" into the local state if the LLO remote message is true. In the Remote state the device is not allowed to respond to local messages which control function that are also controlled by remote messages. A device will exit the remote state and enter the local state when REN goes false. It will also enter the local state if the GTL (go to local) remote message is true and the device has been addressed to listen. It will also enter the local state if the rtl message is true and the LLO message is false or ACDS is inactive.

A device will exit the remote state and enter RWLS (remote with lockout state) if the LLO (local lockout) message is true and ACDS is active. In this mode, those local messages which control functions which are also controlled by remote messages are ignored. In other words, the "rtl" message is ignored. A device will exit RWLS and go to the local state if REN goes false. The device will exit RWLS and go to LWLS if the GTL message is true and the device is addressed to listen.

Polling

The IEEE-488 standard specifies two methods for a slave device to let the controller know that it needs service.

These two methods are called Serial and Parallel Poll. The controller performs one of these two polling methods after a slave device requests service. As implied in the name, a Serial Poll is when the controller sequentially asks each device if it requested service. In a Parallel Poll the controller asks all of the devices on the GPIB if they requested service, and they reply in parallel.

Serial Poll

When the controller performs a Serial Poll, each slave device sends back to the controller a Serial Poll Status Byte. One of the bits in the Serial Poll Status Byte indicates whether this device requested service or not. The remaining 7 bits are used defined, and they are used to indicate what type of service is required. The IEEE-488 spec only defines the service request bit, however HP has defined a few more bits in the Serial Poll Status Byte. This can be seen in Figure 4.

When a slave device needs service it drives the SRQ line on the GPIB bus true (low). For the 8291A this is done by setting bit 7 in the Serial Poll Status Byte. The CPU in the controller may be interrupted by SRQ or it may poll a register to determine the state of SRQ. Using the 8292 one could either poll the interrupt status register for the SRQ interrupt status bit, or enables SRQ to interrupt the CPU. After the controller recognizes a service request, it goes into the serial poll routine.

The first thing the controller does in the serial poll routine is assert ATN. When ATN is asserted true the controller takes control of the GPIB, and all slave de-

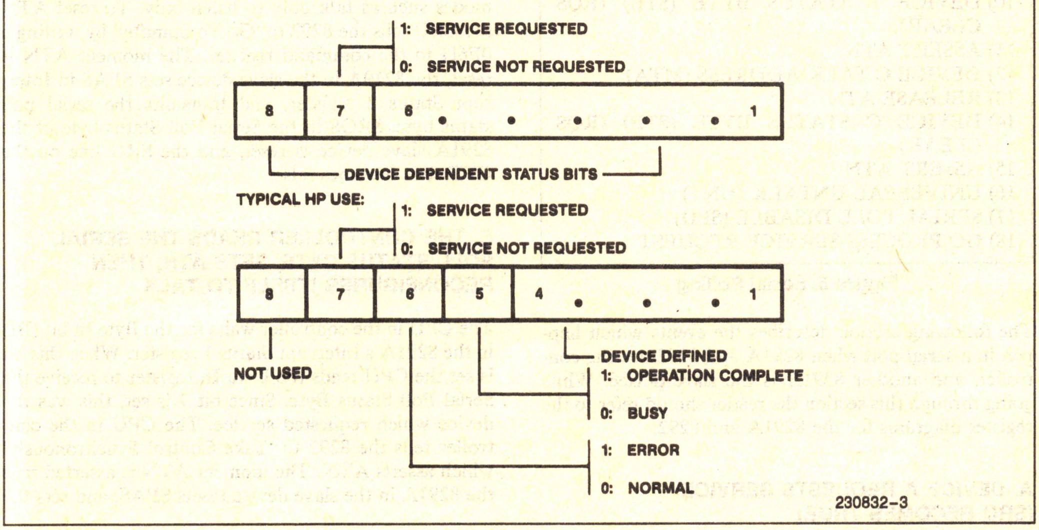


Figure 4. The Serial Poll Status Byte

vices on the bus must listen. All bytes sent over the bus while ATN is true are commands. After the controller takes control, it sends out a Universal Unlisten (UNL), which tells all previously addressed listeners to stop listening. The controller then sends out a byte called SPE (Serial Poll Enable). This command notifies all of the slaves on the bus that the controller has put the GPIB in the Serial Poll Mode State (SPMS). Now the controller addresses the first slave device to TALK and puts itself in the listen mode. When the controller resets ATN the device addressed to talk transmits to the controller its Serial Poll Status Byte. If the device just polled was the one requesting service, the SRQ line on the GPIB goes false, and bit 7 in the serial poll status byte of the 8291A is reset. If more than one device is requesting service, SRQ remains low until all of the devices requesting service have been polled, since SRQ is wire-ored. To continue the Serial Poll, the controller asserts ATN, addresses the next device to talk then reads the Serial Poll Status Byte. When the controller is finished polling it asserts ATN, sends the univeral untalk command (UNT), then sends the Serial Poll Disable command (SPD). The flow of the serial poll can be seen from the example in Figure 5.

0) DEVICE A REQUESTS SERVICE (SRQ)
1) ASSERT ATN
2) UNIVERSAL UNLISTEN (UNL)
3) SERIAL POLL ENABLE (SPE)
4) DEVICE A TALK ADDRESS (MTA)
5) RELEASE ATN
6) DEVICE A STATUS BYTE (STD) (RQS SET)
7) ASSERT ATN
8) DEVICE B TALK ADDRESS (MTA)
9) RELEASE ATN
10) DEVICE B STATUS BYTE (STB) (RQS CLEAR)
11) ASSERT ATN
12) DEVICE C TALK ADDRESS (MTA)
13) RELEASE ATN
14) DEVICE C STATUS BYTE (STB) (RQS CLEAR)
15) ASSERT ATN
16) UNIVERSAL UNTALK (UNT)
17) SERIAL POLL DISABLE (SPD)
18) GO PROCESS SERVICE REQUEST

Figure 5. Serial Polling

The following section describes the events which happen in a serial poll when 8291A and 8292 are the controller, and another 8291A is the slave device. While going through this section the reader should refer to the register diagrams for the 8291A and 8292.

A. DEVICE A REQUESTS SERVICE (SRQ BECOMES TRUE)

The slave devices rsv bit in the 2819A's serial poll mode register is set.

B. CONTROLLER RECOGNIZES SRQ AND ASSERTS ATN

The 8292's SPI pin 33 interrupts the CPU. The CPU reads the 8292's Interrupt status register and finds the SRQ bit set. The CPU tells the 8292 to 'Take Control Synchronously' by writing a OFDH to the 8292's command register.

C. THE CONTROLLER SENDS OUT THE FOLLOWING COMMANDS: UNIVERSAL UNLISTEN (UNL), SERIAL POLL ENABLE (SPE), MY TALK ADDRESS (MTA)

(MTA is a command which tells one of the devices on the bus to talk.)

The CPU in the controller waits for a BO (byte out) interrupts in the 8291A's interrupt status 1 register before it writes to the Data Out register a 3FH (UNL), 18H (SPE), 010XXXXX (MTA). The X represents the programmable address of a device on the GPIB. When the 8291A in the slave device receives its talk address, the ADSC bit in the Interrupt Status register 2 is set, and in the Address Status Register TA and TPAS bits are set.

D. CONTROLLER RECONFIGURES ITSELF TO LISTEN AND RESETS ATN

The CPU in the controller puts the 8291A in the listen only mode by writing a 40H to the Address Mode register of the 8291A, and then a OOH to the Aux Mode register. The second write is an 'Immediate Execute pon' which must be used when switching addressing modes such as talk only to listen only. To reset ATN the CPU tells the 8292 to 'Go To Standby' by writing a 0F6H to the command register. The moment ATN is reset, the 8219A in the slave device sets SPAS in Interrupt Status 2 register, and transmits the serial poll status byte. SRQS in the Serial Poll Status byte of the 8291A slave device is reset, and the SRQ line on the GPIB bus becomes false.

E. THE CONTROLLER READS THE SERIAL POLL STATUS BYTE, SETS ATN, THEN RECONFIGURES ITSELF TO TALK

The CPU in the controller waits for the Byte In bit (BI) in the 8291A's Interrupt Status 1 register. When this bit is set the CPU reads the Data In register to receive the Serial Poll Status Byte. Since bit 7 is set, this was the device which requested service. The CPU in the controller tells the 8292 to 'Take Control Synchronously' which asserts ATN. The moment ATN is asserted true the 8291A in the slave device resets SPAS, and sets the

Serial Poll Complete (SPC) bit in the Interrupt Status 2 register. The controller reconfigures itself to talk by setting the TO bit in the Address Mode register and then writing a OOH to the Aux Mode register.

F. THE CONTROLLER SENDS THE COMMANDS UNIVERSAL UNTALK (UNT), AND SERIAL POLL DISABLE (SPD) THEN RESETS THE SRQ BIT IN THE 8292 INTERRUPT STATUS REGISTER

The CPU in the controller waits for the BO Interrupt status bit to be set in the Interrupt Status 1 register of the 8291A before it writes 5FH (UNT) and 19H (SPD) to the Data Out register. The CPU then writes a 2BH to the 8292's command register to reset the SRQ status bit in the Interrupt Status register. When the 8291A in the slave device receives the UNT command the ADSC bit in the Interrupt Status 2 register is set, and the TA and TPAS bits in the Address Status register will be reset. At this point the controller can service the slave device's request.

Note that in the software listing of AP-66 (USING THE 8292 GPIB CONTROLLER) there is a bug in the serial poll routines. In the 'SRQ ROUTINE' when the CPU finds that the SRQ bit in the interrupt status register is set, it immediately writes the interrupt Acknowledge command to the 8292 to reset this bit. However the SRQ GPIB line will still be driven true until the slave device driving SRQ has been polled. Therefore, the SRQ status bit in the 8292 will become set and latched again, and as a result the SRQ status bit in the 8292 will still be set after the serial poll. The proper time to reset the SRQ bit in the 8292 is after SRQ on the GPIB becomes false.

Parallel Poll

The 8291A supports an additional method for obtaining status from devices known as parallel poll (PPOL). This method limits the controller to a maximum of 8 devices at a time since each device will produce a single bit response on the GPIB data lines. As shown in the state diagrams, there are three basic parallel poll states: PPIS (parallel poll idle state), PPSS (parallel poll standby state), and PPAS (parallel poll active state).

In PPIS, the device's parallel poll function is in the idle state and will not respond to a parallel poll. PPSS is the standby state, a state in which the device will respond to a parallel poll from the controller. The response is initiated by the controller driving both ATN and EOI true simultaneously.

The 8291A state diagram shows a transition from PPIS to PPSS with the "lpe" message. This is a PP2 implementation for a parallel poll. This "lpe" (local poll enable) local message is achieved by writing $011USP_3P_2P_1$ to the Aux Mode Register with U=0. The S bit is the sense bit. If the "ist" (individual status) local message value matches the sense bit, then the 8291A will give a true response to a parallel poll. Bits P_3-P_1 identify which data line is used for a response.

For example, assume the programmer decides that the system containing the 8291A shall participate in parallel poll. The programmer, upon system initialization would write to the Aux Mode Register and reset the U bit and set the S bit plus identify a data line (P_3-P_1 bits). At "pon," the 8291A would not respond true to a parallel poll unless the parallel poll flag is set (via Aux Mode Register command).

When a status condition in the user system occurs and the programmer decides that this condition warrants a true response, then programmers software should set the parallel poll flag. Since the S bit value matches the "ist" (set) condition a true response will be given to all parallel polls.

An additional method of parallel polling reading exists known as a PP1 implementation. In this case the controller sends a PPE (parallel poll enable) message. PPE contains a bit pattern similar to the bit pattern used to program the "lpe" local message. The 8291A will receive this as an undefined command and use it to generate an "lpe" message. Thus the controller is specifying the sense bits and data lies for a response. A PPD (parallel poll disable) message exists which clears the bits $SP_3P_2P_1$ and sets the U bit. This also will be received by the 8291A and used to generate an "lpe" false local message.

The actual sequence of events is as follows. The controller sends a PPC (parallel poll configure) message. This is an undefined command which is received in the CPT register and the handshake is held off. The local CPU reads this bit pattern, decodes it, and sends a VSCMD message to the Aux Mode Register. The controller then sends a ppe message which is also received as an undefined command in the CPT register. The local CPU reads this, decodes it clears the MSB, and writes this to the Aux Mode Register generating the "lpe" message.

The controller then sends ATN and EOI true and the 8291A drives the appropriate data line if the "ist" (parallel poll flag) is true. The controller will then send a PPD (parallel poll disable) message (again, an undefined command). The CPU reads this from the CPT register and uses it to write new "lpe" message (this "lpe" message will be false). The controller then sends a PPU (parallel poll unconfigure) message. Since this is also an undefined command, it goes into the CPT register. When the local CPU decodes this, the CPU should clear the "ist" (parallel poll flag).

APPLICATION EXAMPLES

In the course of developing this application note, two complete and identical; GPIB systems were built. The schematics and block diagrams are contained in Appendix 1. These systems feature an 8088 CPU, 8237 DMA controller, serial I/O (8215a and 8253), RAM, EPROM, and a complete GPIB talker/listener controller. Jumper switches were provided to select between a controller function and a talker/listener function. This system design is based on the design of Intel's SDK-86 prototyping kit and thus shares the same I/O and memory addresses. This system uses the same download software to transfer object files from Intel development systems.

Two Software Drivers

Two software drivers were developed to demonstrate a ton/lon environment. These two programs (BOARD 1 and BOARD 2) are contained in Appendix 2.

In this example, one of the systems (BOARD 1) initially is programmed in talk-only mode and synchronization is achieved by waiting for the listening board to become active. This is sensed by the lack of a GPIB error since a condition of no active listener produces an ERR status condition. Board 1 upon detecting the presence of an active listener transmits a block of 100 bytes from a PROM memory across the bus. The second system (BOARD 2) receives this data and stores it in a buffer, EOI is sent true by the talker (BOARD 1) with the last byte of data. Upon detection of EOI, BOARD 2 switches to the talk only mode while BOARD 1 upon terminal count switches to the listen only mode. BOARD 2 then detects the presence of an active listener and transmits the contents of its buffer back to BOARD 1 which stores this data in the buffer. EOI again is sent with the last byte and BOARD 2 switches back to listen-only. BOARD 1 upon detecting EOI then compares the contents of its buffer with the contents of its PROM to ensure that no data transmission errors occurred. The process then repeats itself.

8291A with HP 9835A

An example of the 8291A used in conjunction with a bus controller is also included in this application note. In this example, the 8291A system used in previous experiments was connected via the GPIB to a Hewlett-Packard 9835A desktop computer. This computer contains, in addition to a GPIB interface, a black and white CRT, keyboard, tape drive for high quality data cassettes, and a calculator type printer. The software for the HP9835S is shown in Appendix 3. The user should refer to the operation manuals for the HP 9835A for information on the features and programming methods for the HP 9835A.

In this example, the 8292 was removed from its socket and the OPTA and OPTB pins of the two 8293 transceiver reconfigured to modes 0 and 1. Optionally, the mode pins could have been left wired for modes 2 and 3 and the 8292 left in its socket with its SYC pin wired to ground. This would have produced the same effect.

The first action performed is sending IFC. Generally, this is done when a controller first comes on line. This pulse is at least 100 μs in duration as specified by the IEEE-488 standard.

The software checks to see if active listeners are on line. For demonstration purposes, the HP 9835A will flag the operator to indicate that listeners are on line.

The HP 9835A then configures and performs a parallel poll (PPOL). The parallel poll indicates 1 bit of status of each device in a group of up to 8 devices. Such information could be used by an application program to determine whether optional devices are part of a system configuration. Such optional devices might include mass storage devices, printers, etc., where the application software for the controller might need to format data to match each type of device. Once the PPOL sequence is finished, the HP 9835A offers the user the opportunity to execute user commands from the keyboard. At this time the HP 9835A sits in a loop waiting for an SRQ condition. When the operator hits a key on the keyboard, the HP 9835A processor is interrupted and vectors to a service routine where the key is read and the appropriate routine is executed. The HP 9835A will then return to the loop checking for the SRQ true. For this application, the valid keys are G, D, R, H, and X. Pressing the "G" key causes the GET command to be sent across the bus. A message to this effect is printed in the CRT and the HP 9835A returns. The "D" key causes the SDC message to be sent with the 8291A being the addressed device. Again, an appropriate message is output on the HP 9835A CRT. The "R" key causes the GTL message to be sent. The CRT displays "REMOTE MESSAGE SENT." The "H" key causes a menu to be displayed on the HP 9835A CRT screen. This menu lists the allowed commands and their functions. NO GPIB commands are sent. The "X" key allows the operator to send one line of data across the bus. The line of data is terminated by a carriage return and line feed produced by pressing the "CONTINUE" key on the HP 9835A.

The characters are stored in the sequence entered into a buffer whose maximum size is 80 characters. Pressing the "CONTINUE" key terminates storing characters in the array and all characters including the carriage return and line feed are sent. EOI is then sent true with a false byte of OOH. This false byte is due to the 1975 standard which allows asynchronous sending and reception of EOI. (The 8291A supports the later 1978 standard which eliminates this false byte.)

After any key command is serviced control returns to the loop which checks for SRQ active. Should SRQ be active, then the keyboard interrupt is disabled and a message printed to indicate that SRQ has been received true.

The controller then performs a parallel poll.

This is an example of how parallel poll may be used to quickly check which group of devices contains a device sending SRQ. The eight devices in a group would, of course, have software drivers which allow a true response to a PPOL if that device is currently driving SRQ true. This would be a valuable method of isolation of the SRQ source in a system with a large number of devices. In this application program, only the response from the 8291A is of concern and only the 8291A's response is considered. It does, however, demonstrate the technique employed. If a true response from the 8291A is detected, then a message to this effect is printed on the HP 9835A CRT screen. From this process, the controller has identified the device requesting service and will use a serial poll (SPOL) to determine the reason for the service request. This method of using PPOL is not specifically defined by the IEEE-488 standard but is a use of the resources provided.

The controller software then prints a message to indicate that it is about to perform a serial poll. This serial poll will return to the controller the current status of the 2819A and clear the service request. The status byte received is then printed on the CRT screen of the HP 9835A. One of the 8291A status bits indicates that the 8291A system has a field (on line or less) of data to transfer to the HP 9835A. If this bit is set, then the HP 9835A addresses the 8291A system to talk. The data is sent by the 8291A system is then printed on the CRT screen of the HP 9835A. The HP 9835 then enables the keyboard interrupts and goes into its SRQ checking loop.

Appendix 4 contains the software for the 8291A system which is connected to the HP 9835A via the GPIB. This software throws away the first byte of data it receives since this transfer was used by the HP 9835A to test when the 8291A system came on line.

Next, both status registers are read and stored in the two variable STAT 1 and STAT 2. It is necessary to store the status since reading the status registers clears the status bits.

Initially, six status bits are evaluated (END, GET, CPT, DEC, REMC, ADSC). Some of these conditions require that additional status bits be evaluated.

If END is true, then the 8291A system has received a block from the HP 9835A and the contents of a buffer is printed on the CRT screen. Next, the CPT bit is checked. PPC and PPE are only valid undefined commands in this example.

Next, the GET bit is examined and if true, the CRT screen connected to the serial channel on the 8291A system prints a message to indicate that the trigger command has been received. A similar process occurs with the DEC and REMC status bits.

Address Status Chagne (ADSC) is checked to see if the 8291A has been addressed or unaddressed by the controller. If ADSC is false, then the software checks the keyboard at the CRT terminal. If ADSC is set, then the TA and LA bits are read and evaluated to determine whether the 8291A has been addressed to talk or listen. The DMA controller is set to start transfers at the start of the character buffer and the type of transfer is determined by whether the 8291A in in TADS or LADS. We only need to set up the DMA controller since the transfers will be transparent to the system processor. The keyboard from the CRT terminal is then checked. If a key has been hit, then this character is stored in the character buffer and the buffer printer set to the next character location. This process repeats until the received character is a line feed. The line feed is echoed to the CRT, the serial poll status byte updated and the SRQ line driven true. This allows the 8291A system to store up to one line of characters before requesting a transfer to the controller. Recall that upon receiving an SRQ, the controller will perform a serial poll and subsequently address the 8291A to talk. The 8291A system then goes back to reading the status register thus repeating the process.

CONCLUSION

This application note has shown a basic method to view the IEEE 488 bus, when used in conjunction with Intel's 8291A.

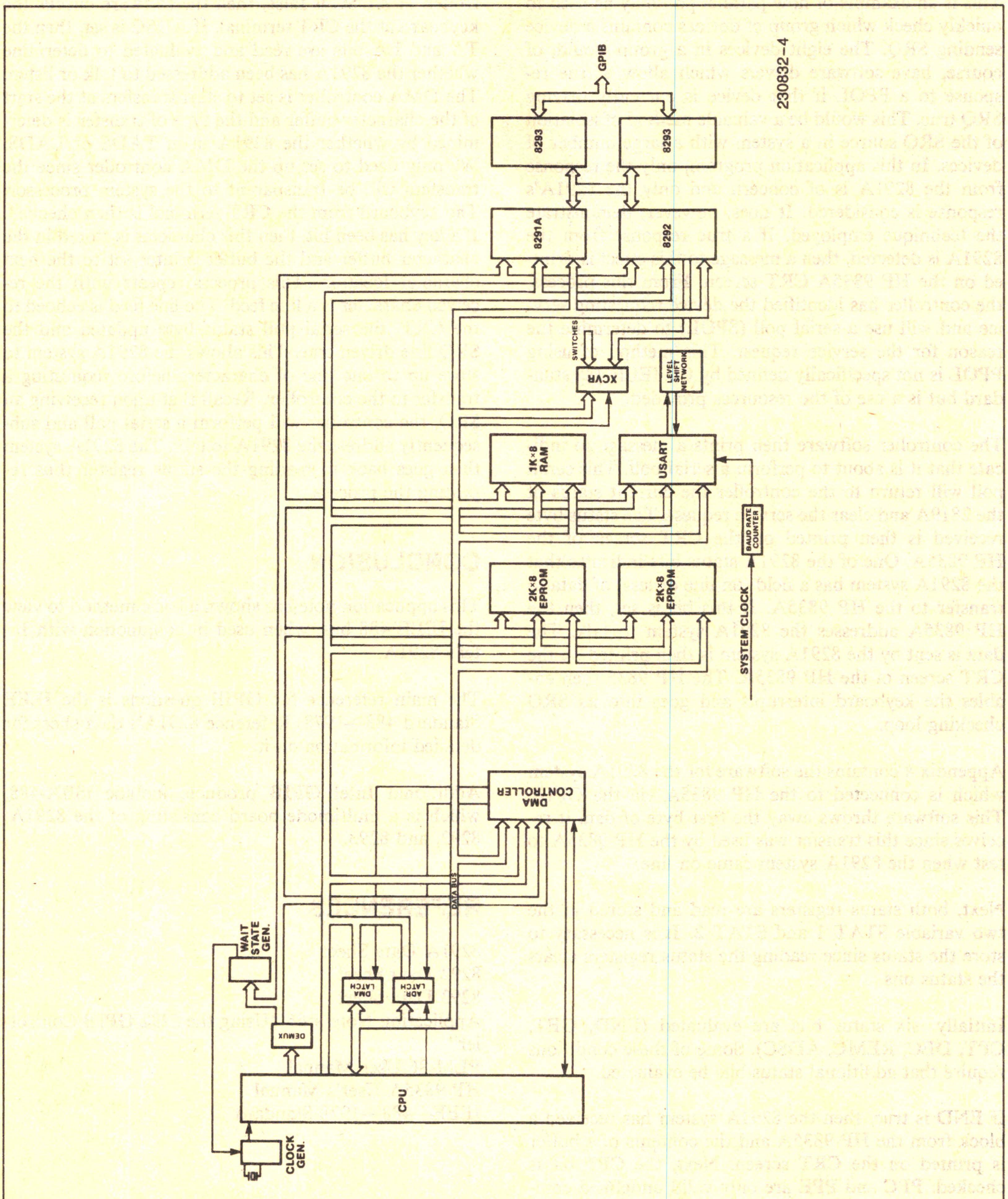
The main reference for GPIB questions is the IEEE Standard 488—1978. Reference 8291A's data sheet for detailed information on it.

Additional Intel GPIB products include iSBX-488, which is a multimode board consisting of the 8291A, 8292, and 8293.

REFERENCES

8291A Data Sheet
8292 Data Sheet
8293 Data Sheet
Application Note #66 "Using the 8292 GPIB Controller"
PLM-86 User Manual
HP 9835A User's Manual
IEEE—488—1978 Standard

APPENDIX A
SYSTEM BLOCK DIAGRAM WITH 8088



APPENDIX B
SOFTWARE DRIVERS FOR BLOCK DATA TRANSFER

```
PL/M-86 COMPILER    BOARD 1

ISIS-II PL/M-86  V1.1  COMPILATION OF MODULE BOARD 1
OBJECT MODULE PLACED IN: F1. BRD1  OBJ
COMPILER INVOKED BY:    PLM86.  F1:  BRD1.  SRC SYMBOLS MEDIUM

            /*   BOARD 1 TPT PROGRAM            */
            /*   THIS BOARD TALKS TO THE OTHER BOARD BY   */
            /*   TRANSFERRING A BLOCK OF DATA VIA THE 8237  */
            /*   COUPLED WITH THE 8291A.   THE 8291A IS PROGRAM- */
            /*   MED TO SEND EOI WHEN RECOGNIZING THE LAST    */
            /*   DATA BYTE'S BIT PATTERN.  WHILE DATA IS BEING */
            /*   TRANSFERRED, THE PROCESSOR PERFORMS  I/O READS */
            /*   OF THE 8237 COUNT REGISTERS TO SIMULATE BUS   */
            /*   ACTIVITY, AND TO DETERMINE WHEN TO TURN THE   */
            /*   LINE AROUND. AFTER THE 8237 HAS REACHED  */
            /*   TERMINAL COUNT, THE 8291A IS PROGRAMMED TO     */
            /*   THE LISTENER STATE AND WAITS FOR THE BLOCK    */
            /*   TO BE TRANSMITTED BACK FROM THE SECOND BOARD. */
            /*   THIS DATA IS PLACED IN A SECOND BUFFER AND    */
            /*   ITS CONTENTS COMPARED WITH THE ORIGINAL DATA  */
            /*   TO CHECK FOR INTERFACE INTEGRITY.   */

1           BOARD1:

            DO;
                /* PROCEDURES */

2   1           CO:  PROCEDURE (XXX) ;
3   2                   DECLARE XXX BYTE,
                        SER$STAT  LITERALLY  'OFFF2H',
                        SER$DATA  LITERALLY  'OFFFOH',
                        TXRDY     LITERALLY  '01H';
4   2                   DO WHILE (INPUT  (SER$STAT) AND TXRDY) <>     TXRDY;
5   3                   END;
6   2                   OUTPUT (SER$DATA)       = XXX;
7   2               END CO;

                    /*   SETUP BUFFERS */
8   1                   DECLARE BUFF2 (100)   BYTE;    /*  RAM STORAGE AREA */
9   1                       DECLARE BUFF1 (100) BYTE DATA

                        (1,2,3,4,5,6,7,8,9,10H,
                        11H, 12H, 13H, 14H, 15H, 16H, 17H, 18H, 19H, 20H,
                        21H, 22H, 23H, 24H, 25H, 26H, 27H, 28H, 29H, 30H,
                        31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H, 40H,
                        41H, 42H, 43H, 44H, 45H, 46H, 47H, 48H, 49H, 50H,
                        51H, 52H, 53H, 54H, 55H, 56H, 57H, 58H, 59H, 60H,
                        61H, 62H, 63H, 64H, 65H, 66H, 67H, 68H, 69H, 70H,
                        71H, 72H, 73H, 74H, 75H, 76H, 77H, 78H, 79H, 80H,
                        81H, 82H, 83H, 84H, 85H, 86H, 87H, 88H, 89H, 90H,
```

230832-5

```
PL/M-86  COMPILER   BOARD1

                    91H, 92H, 93H, 94H, 95H, 96H, 97H, 98H, 99H, ODH);
10   1              DECLARE BUFF3(17)  BYTE DATA
                    (ODH, OAH, 'COMPARE ERROR', ODH, OAH);  /* ROM STORAGE AREA */

                    /* 8237 PORT ADDRESSES   */

11   1              DECLARE

                    CLEAR$FF          LITERALLY 'OFFDDH',  /* MASTER CLEAR */
                    START$0$LO        LITERALLY 'OFFDOH',
                    START$0$HI         LITERALLY 'OFFDOH',
                    0$COUNT$LO        LITERALLY 'OFFD1H',
                    0$COUNT$HI        LITERALLY 'OFFD1H',
                    SET$MODE          LITERALLY 'OFFDBH',
                    CMD$37            LITERALLY 'OFFD8H',
                    SET$MASK          LITERALLY 'OFFDFH';

                    /* 8237 COMMAND   - DATA BYTES */
12   1              DECLARE           DMA$ADR$TALK    POINTER;
13   1              DECLARE           DMA$ADR$LSTN    POINTER;

14   1              DECLARE

                    RD$TRANSFER       LITERALLY '48H',
                    WR$TRANSFER       LITERALLY '44H',
                    NORM$TIME         LITERALLY '20H',
                    TC$LO1            LITERALLY 'OFFH',
                    TC$HI1            LITERALLY '00H',
                    TC$LO2            LITERALLY '99D',           /* 100 XFERS   */
                    TC          LITERALLY '01H',
                    I                 BYTE;

15   1              DECLARE

                    DMA$WRD$TALK (2)   WORD    AT    (@DMA$ADR$TALK),
                    DMA$WRD$LSTN(2)   WORD    AT    (@DMA$ADR$LSTN);

                    /* 8291A PORT ADDRESSES */

16   1              DECLARE

                    PORT$OUT          LITERALLY 'OFFCOH',  /* DATA OUT*/
                    PORT$IN           LITERALLY 'OFFCOH'
                    STATUS$1          LITERALLY 'OFFC1H',  /*INTR STAT 2*/
                    STATUS$2          LITERALLY 'OFFC2H',  /* INTR STAT 2 */
                    ADDR$STATUS       LITERALLY 'OFFC4H',
                    COMMAND$MOD       LITERALLY 'OFFC5H',  /*CMD PASS THRU */
                    ADDR$0            LITERALLY 'OFFC6H',
                    EOS$REG           LITERALLY 'OFFC7H',  /*   EOS REGISTER */
```

230832-6

```
              /* 8291A COMMAND  - DATA BYTES */
PL/M-86 COMPILER    BOARD1

17  1                    DECLARE

                END$EOI   LITERALLY        '88H',
                DNE  LITERALLY        '10H',
                PON  LITERALLY        '00H',
                RESET     LITERALLY        '02H',
                CLEAR     LITERALLY        '00H',
                DMA$REQ$L LITERALLY        '10H',
                DMA$REQ$T LITERALLY        '20H',
                MOD1$TO   LITERALLY        '80H',
                MOD1$LO   LITERALLY        '40H',
                EOS  LITERALLY        '0DH',
                PRESCALER LITERALLY        '23H',
                HIGH$SPEED LITERALLY        '0A4H',
                OKAY      LITERALLY        '0FFFFH',
                XYZ  BYTE,
                MATCH     WORD,
                BO   LITERALLY        '02H',
                BI   LITERALLY        '01H',
                ERR  LITERALLY        '04H';

          /* CODE BEGINS */

18  1     START91:

                OUTPUT  (STATUS$2)  =CLEAR;  /* SHUT-OFF DMA REQ BITS TO */
                                             /* PREVENT EXTRA DMA REQS   */
                                              /*FROM 8291A               */

                /* MANIPULATE DMA ADDRESS VARIABLES */

19  1           DMA$ADR$TALK    =(@BUFF1);
20  1           DMA$ADR$LSTN    =(@BUFF2);
21  1           DMA$WRD$TALK(1)=SHL  (DMA$WRD$TALK(1), 4);
22  1           DMA$WRD$TALK(0)=DMA$WRD$TALK (0) + DMA$WRD$TALK (1);
23  1           DMA$WRD$LSTN(1)=SHL  (DMA$WRD$LSTN (1), 4);
24  1           DMA$WRD$LSTN(0)=DMA$WRD$LSTN (0) +DMA$WRD$LSTN (1);

25  1        INIT37T:
            /* INIT 8237 FOR TALKER FUNCTIONS */

26  1           OUTPUT     (CLEAR$FF)      =CLEAR; /* TOGGLE MASTER CLEAR */
27  1           OUTPUT     (CMD$37)        =NORM$TIME;
28  1           OUTPUT     (SET$MODE)      =RD$TRANSFER;
                OUTPUT     (SET$MASK)      =CLEAR;
29  1           OUTPUT     (START$0$LO)      =DMA$WRD$TALK (0);
30  1           DMA$WRD$TALK (0)           =SHR (DMA$WRD$TALK (0), 8);
31  1           OUTPUT     (START$0$HI)      =DMA$WRD$TALK (0);
32  1           OUTPUT     (0$COUNT$LO)      =TC$LO2;
33  1           OUTPUT     (0$COUNT$HI)      =TC$HI2;
                /* INIT 8291A FOR TALKER FUNCTIONS */

PL/M-86 COMPILER     BOARD1
```

230832-7

```
34   1          OUTPUT   (EOS$REG)        =EOS;
35   1          OUTPUT   (COMMAND$MOD)        =END$EOI;  /*  EOI ON EOS SENT */
36   1          OUTPUT   (ADDR$STATUS)        =MOD1$TO;  /* TALK ONLY */
37   1          OUTPUT   (COMMAND$MOD)        =PRESCALER,
38   1          OUTPUT   (COMMAND$MOD)        =HIGH$SPEED;
39   1          OUTPUT   (COMMAND$MOD)        =PON;

40   1          DO WHILE (INPUT  (STATUS$1) AND BO)  =0;
41   2          END;    /*  WAIT FOR BO INTR   */
42   1          OUTPUT   (PORT$OUT)  = OAAH;

43   1          DO WHILE  (INPUT  (STATUS$1) AND ERR) = ERR;
44   2             DO WHILE  (INPUT  (STATUS$1)  AND BO) = O;
45   3             END;  /*  WAIT FOR BO INTR   */
46   2             OUTPUT  (PORT$OUT)   =OAAH;
47   2          END;

48   1          OUTPUT  (STATUS$2)   =DMA$REQ$T;   /*  ENABLE DMA REQS */

49   1              DO WHILE  (INPUT  (CMD$37)  AND TC) <>        TC;
                     /* WAIT FOR TC  = 0 */
50   2              END;

51   1          INIT37L;

                OUTPUT  (STATUS$2)   =CLEAR;   /*  DISABLE DMA REQS */

                /* INIT 8237 FOR LISTENER FUNCTIONS */

52   1          OUTPUT   (CLEAR$FF) O=CLEAR; /*  TOGGLE MASTER RESET */
53   2          OUTPUT   (CMD$37)     =NORM$TIME;
54   1          OUTPUT   (SET$MODE)   =WR$TRANSFER;
55   1          OUTPUT   (SET$MASK)   =CLEAR;
56   1          OUTPUT   (START$O$LO)    =DMA$WRD$LSTN (O);
57   1          DMA$WRD$LSTN (O)      =SHR (DMA$WRD$LSTN (O), 8);
58   1
      OUTPUT        (START $O$HI)      =DMA$WRD$LSTN (O);
59   1          OUTPUT   (O$COUNT$LO)        =TC$LO1;
60   1          OUTPUT   (O$COUNT$HI)        =TC$HI1;

                /* INIT 8291A FOR LISTENER FUNCTIONS   */

61   1          OUTPUT   (COMMAND$MOD)        =RESET,
62   1          OUTPUT   (ADDR$STATUS)        =MOD1$LO;  /*  LISTEN ONLY */
63   1          OUTPUT   (COMMAND$MOD)        =PON;

64   1          DO WHILE (INPUT  (STATUS$1)   AND BI)   =0;
65   2          END;  /*  WAIT FOR BI INTR */
66   1          XYZ  = INPUT (PORT$IN);

67   1          OUTPUT  (STATUS$2)   =DMA$REQ$L;   /*  ENABLE DMA REQS   */

68   1          DO WHILE  (INPUT  (STATUS$1)  AND DNE)<>        DNE;
                /*  WAIT FOR EOI RECEIVED  */
```

230832-8

```
PL/M-86 COMPILER    BOARD 1

         70   1      CMPBLKS

                            /* COMPARE THE TWO BUFFERS CONTENTS */

                            MATCH=CMPB  (@BUFF1, @BUFF2, 100);

         71   1             IF MATCH = OKAY THEN GOTO START91;

                            /*  SEND ERROR MESSAGE IN BUFFER 3  */

         73   1             DO I=0 TO 16;
         74   2                 CALL CO  (BUFF 3  (I)  );
         75   2             END;

         76   1             GOTO START91;

         77   1             END;

MODULE INFORMATION:

         CODE AREA SIZE           =01DBH    475D
         CONSTANT AREA SIZE       =0075H    117D
         VARIABLE AREA SIZE       =0070H    112D
         MAXIMUM STACK SIZE       =0006H      6D
         243 LINES READ
         0 PROGRAM ERROR (S)

END OF PL/M-86  COMPILATION
```

230832-9

```
PL/M-86 COMPILER    BOARD2

ISIS-II PL/M-86  V1.1 COMPILATION OF MODULE BOARD2
OBJECT MODULE PLACED IN :F1:BRD2.OBJ
COMPILER INVOKED BY:    PLM86 :F1:BRD2.SRC

                         /* BOARD 2 TPT PROGRAM          */
                         /*        */
                         /* THIS BOARD LISTENS TO THE OTHER BOARD (1)    */
                         /* AND DMA'S DATA INTO A BUFFER, WHILE WAITING */
                         /* FOR THE END INTERRUPT BIT TO BECOME ACTIVE  */
                         /* UPON END ACTIVE, THE DATA IN THE BUFFER IS  */
                         /* SENT BACK TO THE FIRST BOARD VIA THE GPID   */
                         /* WHEN THE BLOCK IS FINISHED THE 8291A IS  */
                         /* PROGRAMMED BACK INTO THE LISTENER MODE    */

     1             BOARD2:

                   DO;
                         /* 8237 PORT ADDRESSES */

  2  1                   DECLARE

                         CLEAR$FF        LITERALLY       'OFFDDH',      /*MASTER CLEAR */
                         START$O$Lo      LITERALLY       'OFFDOH',
                         START$O$HI      LITERALLY       'OFFDOH',
                         O$COUNT$LO      LITERALLY       'OFFD1H',
                         O$COUNT$HI      LITERALLY       'OFFD1H',
                         SET$MODE        LITERALLY       'OFFDBH',
                         CMD$37          LITERALLY       'OFFD8H',
                         SET$MASK        LITERALLY       'OFFDFH',

                         /* 8237 COMMAND  - DATA BYTES  */

  3  1                   DECLARE

                         RD$TRANSFER     LITERALLY       '48H',
                         WR$TRANSFER     LITERALLY       '44H',
                         ADDR$1A         LITERALLY       '00H',
                         ADDR$1B         LITERALLY       '01H',
                         NORM$TIME       LITERALLY       '20H',
                         TC$LO1          LITERALLY       'OFFH',
                         TC$HI1          LITERALLY       '00H',
                         TC$LO2          LITERALLY       '99D',
                         TC$HI2          LITERALLY       '00H',
                         TC         LITERALLY       '01H',

                         /* 8291A  PORT ADDRESSES */

  4  1                   DECLARE

                         PORT$OUT        LITERALLY       'OFFCOH',
                         PORT$IN         LITERALLY       'OFFCOH',/* DATA IN */
                         STATUS$1        LITERALLY       'OFFC1H', /* INTR STAT 1 */
                         STATUS$2        LITERALLY       'OFFC2H', /* INTR STAT 2 */
                         ADDR$STATUS     LITERALLY       'OFFC4H', /* ADDR STAT   */
                         COMMAND$MOD     LITERALLY       'OFFC5H', /* CMD PASS THRU */
```

230832-10

```
PL/M-86 COMPILER   BOARD2

                    ADDR$0     LITERALLY       'OFFC6H',
                    EOS$REG    LITERALLY       'OFFC7H',  /*  EOS REGISTER */

                    /* 8291A  COMMAND - DATA BYTES */

5    1              DECLARE

                    END$EOI    LITERALLY       '88H',
                    DNE  LITERALLY        '10H',
                    PON  LITERALLY        '00H',
                    RESET      LITERALLY       '02H',
                    CLEAR      LITERALLY       '00H',
                    DMA$REQ$L  LITERALLY       '10H',
                    DMA$REQ$T  LITERALLY       '20H',
                    MOD1$TO    LITERALLY       '80H',
                    MOD1$LO    LITERALLY       '40',
                    EOS  LITERALLY        '0DH',
                    PRESCALER LITERALLY        '23H',
                    HIGH$SPEED LITERALLY       'A4H',
                    XYZ        BYTE,
                    BO         LITERALLY '02H',
                    BI         LITERALLY '01H',
                    ERR        LITERALLY '04H',

6    1          START91;

                    OUTPUT   (STATUS$2)   =CLEAR;   /*  END INITILIZATION STATE */

                    /* INIT 8237 FOR LISTENER FUNCTION   */

7    1          INIT37L;

                    OUTPUT   (CLEAR$FF)   =CLEAR; /*  TOGGLE MASTER RESET   */
8    1              OUTPUT   (CMD$37)     =NORM$TIME;
9    1              OUTPUT   (SET$MODE)   =WR$TRANSFER;   /*  BLOCK XFER MODE   */
10   1              OUTPUT   (SET$MASK)   =CLEAR;
11   1              OUTPUT   (START$0$LO)       =ADDR$1A;
12   1              OUTPUT   (START$0$HI)       =ADDR$1B;
13   1              OUTPUT   (O$COUNT$LO)       =TC$LO1;
14   1              OUTPLUT  (O$COUNT$HI)       =TC$HI1;

                    /*   INIT 8291A   FOR LISTENER FUNCTIONS     */

15   1              OUTPUT   (COMMAND$MOD)      =RESET;
16   1              OUTPUT   (ADDR$STATUS)      =MOD1$LO;
17   1              OUTPUT   (COMMAND$MOD)      =PON;
18   1              DO WHILE (INPUT   (STATUS$1)   AND BI)   =0;
19   2              END;   /*   WAIT FOR BI INTR */
20   1              XYZ= INPUT   (PORT$IN);
21   1              OUTPUT    (STATUS$2)        =DMA$REQ$L;

                    /* WAIT UNTIL EOI RCVD AND END INTR-BIT SET   */

22   1                 DO WHILE   (INPUT   (STATUS$1)   AND DNE )  <>   DNE;
```

230832-11

```
PL/M-86   COMPILER   BOARD2

23   1                    END;

24   1              INIT37T:
                      /* INIT 8237 FOR TALKER FUNCTION   */

                    OUTPUT   (STATUS$2)    =CLEAR;     /* CLEAR  8291A  DRQ   */
25   1              OUTPUT   (CLEAR$FF)    =CLEAR;
26   1              OUTPUT   (CMD$37)      =NORM$TIME;
27   1              OUTPUT   (SET$MODE)    =RD$TRANSFER;   /*  BLOCK XFER MODE */
28   1              OUTPUT   (SET$MASK)    =CLEAR;
29   1              OUTPUT   (START$0$LO)       =ADDR$1A;
30   1              OUTPUT   (START$0$HI)       =ADDR$1B;
31   1              OUTPUT   (0$COUNT$LO)       =TC$LO2;
32   1              OUTPUT   (0$COUNT$HI)       =TC$HI2;

                    /*  INIT 8291A  FOR TALKER FUNCTION   */

33   1              OUTPUT   (EOS$REG)      =EOS;
34   1              OUTPUT   (COMMAND$MOD)          =END$EOI;/* EOI ON EOS SENT   */
35   1              OUTPUT   (ADDR$STATUS)          =MOD1$TO;/* TALK ONLY   */
36   1              OUTPUT   (COMMAND$MOD)          =PRESCALER;
37   1              OUTPUT   (COMMAND$MOD)          =HIGH$SPEED;
38   1              OUTPUT   (COMMAND$MOD)          =PON;

39   1              DO WHILE  (INPUT  (STATUS$1)   AND BO)    =0;
40   2              END;     /* WAIT FOR BO INTR   */
41   1              OUTPUT  (PORT$OUT)     =0AAH;

42   1              DO WHILE  (INPUT  (STATUS$1)  AND ERR)    =ERR;
43   2                DO WHILE   (INPUT   (STATUS$1)   AND BO)    =0;
44   3                END;     /* WAIT FOR BO INTR   */
45   2                OUTPUT  (PORT$OUT)   =0AAH;
46   2              END;

47   1              OUTPUT   (STATUS$2)     =DMA$REQ$T;
                    /*  WAIT FOR TC=0   */

48   1                  DO WHILE    (INPUT   (CMD$37)  AND TC) <> TC;
49   2                  END;

50   1              GOTO START91;

51   1        END;

MODULE INFORMATION

     CODE AREA SIZE        =0122H     290D
     CONSTANT AREA SIZE    =0000H       0D
     VARIABLE AREA SIZE    =0001H       1D
     MAXIMUM STACK SIZE    =0000H       0D
     152 LINES READ
     0 PROGRAM ERROR (S)
```

230832-12

APPENDIX C
SOFTWARE FOR HP 9835A

```
10   REM SEND IN
TERFACE CLEAR
20   ABORTIO 7
30   REM FORCE E
RRORS UNTIL LIST
ENERS ACTIVE
40 Frcerr:   OUT
PUT 704 USING "#
,K";"B"
50 Chkstst:   ST
ATUS 7;Stat1,Sta
t2,Stat3,Stat4
60   Err=Stat2 A
ND 1
70   IF Err=1 TH
EN GOTO Frcerr
80   PRINT CHR$(
12),"LISTENERS A
RE ON LINE "
90   REM CONFIGU
RE PPOLL
100  PPOLL CONFI
GURE 704;"000001
00"
110

   ! response on
 bit 4
120  PRINT CHR$(
12),"PARALLEL PO
LL CONFIGURED"
130  REM ENABLE
KEYBOARD INTERRU
PT
140  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
150 Keyen:  ON K
BD GOSUB 610
160  STATUS 7;St
at1,Stat2,Stat3,
Stat4
170  Sra=BINAND(
Stat1,128)
180  IF Sra=0 TH
EN GOTO Keyen
190  OFF KBD
200  PRINT CHR$(
12),"SRQ RECEIVE
D"
210  PRINT "SEND
ING PARALLEL POL
L RESPONSE MESSA
GE"
220  REM EXECUTI
NG PARALLEL POLL
230  Ppollbyte=P
POLL(7)
240  PRINT "PARA
LLEL POLL BYTE =
 ";Ppollbyte
250  PRINT "----
----------------
------"
260  Ppollbyte=B
INAND(Ppollbyte,
8)
270  IF Ppollbyt
e=0 THEN GOTO P8
291
280  PRINT "SRQ
NOT FROM 8291"
281  PRINT "COMM
AND = ?    (HIT
'H' FOR LIST)"
290  GOTO Keyen
300 P8291:  PRIN
T "SRQ IS FROM N
CC 8291 ... THE
ENTERPRISE"
310  PRINT "PERF
```

230832-13

```
ORMING SERIAL PO
LL TO GET STATUS
 "
320  STATUS 704;
Stat
330  PRINT CHR$(
12),"Status = ";
Stat
340  Dxfer=BINAN
D(Stat,1)
520  IF Dxfer>0
THEN GOTO Rcvr
530  GOTO Keyen
531 Rcvr:  REM R
EADY TO RCV CHAR
S FROM GPIB
540  DIM G$[80]
550  ENTER 704 U
SING "%,T";G$
560  PRINT CHR$(
12),G$
570  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
580  GOTO Keyen
590  REM INTERRU
PT SERVICE ROUTI
NES
600  REM GET KEY
BOARD DATA
610 Whatkey:  DI
M K$[80]
620  K$=KBD$
630  IF K$="G" T
HEN GOTO Get
640  IF K$="D" T
HEN GOTO Dec
650  IF K$="R" T
HEN GOTO Rem
660  IF K$="H" T
HEN GOTO Help
670  IF K$="X" T
HEN GOTO Xmit
680 Get:  TRIGGE
R 704
690  PRINT CHR$(
12),"GROUP EXECU
TE TRIGGER SENT"
700  PRINT " "
710  PRINT "COMM
AND = ?    (HIT
'H' FOR LIST)"
720  RETURN
730 Dec:  RESET
704
740  PRINT CHR$(
12),"SELECTIVE D
EVICE CLEAR SENT
"
750  PRINT " "
760  PRINT "COMM
AND = ?    (HIT
'H' FOR LIST)"
770  RETURN
780 Rem:  LOCAL
704
790  PRINT CHR$(
12),"REMOTE MESS
AGE SENT"
800  PRINT " "
810  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
820  RETURN
830 Help:  PRINT
 CHR$(12)
840  PRINT " @@@
@ OPERATOR ALLOW
ABLE COMMANDS @@
@@ "
850  PRINT " hit
 key   result"
860  PRINT "    G
       Send GET m
essage"
870  PRINT "    D
       Send DEC m
essage"
```

230832-14

```
880  PRINT "    R
       Send REM/L
OC message"
890  PRINT "    X
       Xmits keyb
oard input to 82
91"
900  PRINT "    H
       Prints thi
s table"
910  PRINT " "
920  PRINT "...
go ahead, TRY IT
 !"
930  RETURN
940 Xmit:  DIM A
$[80]
950  PRINT CHR$(
12),"Enter data
to send and hit
CONTINUE"
960  INPUT A$
970  OUTPUT 704;
A$
971  EOI 7;0
980  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
990  RETURN
1000 END
```

230832-15

APPENDIX D
SOFTWARE FOR HP 8088/HP 9835A VIA GPIB

```
PL/M-86 COMPILER    HPIB

ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE HPIB
OBJECT MODULE PLACED IN :F1:HPIB.OBJ
COMPILER INVOKED BY:  PLM86 :F1:HPIB.SRC LARGE


 1          HPIB:
            /*

            PARAMETER DECLARATIONS
            */

            DO;

 2    1     DECLARE

            ADDR$HI      LITERALLY   '01H',
            ADDR$LO      LITERALLY   '00H',
            ADSC         LITERALLY   '01H',
            BI      LITERALLY   '01H',
            BO      LITERALLY   '02H',
            CHAR$COUNT   BYTE,
            CHAR         BYTE,
            CHARS(80)    BYTE,
            CLEAR        LITERALLY   '00H',
            CPT     LITERALLY   '80H',
            CRLF         LITERALLY   '0AH',
            DEC     LITERALLY   '08H',
            DMA$ADR$LSTN     POINTER,
            DMA$ADR$TALK     POINTER,
            DMA$WRD$LSTN(2) WORD AT      (@DMA$ADR$LSTN),
            DMA$WRD$TALK(2) WORD AT      (@DMA$ADR$TALK),
            DMA$REQ$L    LITERALLY   '10H',
            DMA$REQ$T    LITERALLY   '20H',
            DNE     LITERALLY   '10H',
            END$EOI      LITERALLY   '88H',
            EOS     LITERALLY   '0DH',
            ERR     LITERALLY   '04H',
            GET     LITERALLY   '20H',
            I       BYTE,
            LISTEN       LITERALLY   '04H',
            MLA     LITERALLY   '04H',
            MODE$1       LITERALLY   '01H',
            NO$DMA       LITERALLY   '00H',
            NO$RSV       LITERALLY   '00H',
            NORM$TIME    LITERALLY   '20H',
            PON     LITERALLY   '00H',
            PPC     LITERALLY   '05H',
            PPE$MASK     LITERALLY   '60H',
            PPOLL$CNFG$FLAG LITERALLY    '01H',
            PPOLL$EN$BYTE    BYTE,
            PRI$BUF(80) BYTE AT      (@CHARS),
            RD$XFER      LITERALLY   '48H',
            RESET        LITERALLY   '02H',
            REMC         LITERALLY   '02H',
            RSV     LITERALLY   '40H',
            RXRDY        LITERALLY   '02H',
```

230832-16

```
PL/M-86 COMPILER    HPIB

                SRGS           LITERALLY    '40H',
                STAT1          BYTE,
                STAT2          BYTE,
                TALK           LITERALLY    '02H',
                TA$OR$LA       BYTE,
                TRQ       LITERALLY    '41H',
                TC        LITERALLY    '01H',
                TC$HI          LITERALLY    '00H',
                TC$LO          LITERALLY    '0FFH',
                TXRDY          LITERALLY    '01H',
                UDC       BYTE,
                WR$XFER        LITERALLY    '44H',
                XYZ       BYTE;

                /*

                PORT DECLARATIONS

                */

  3    1        DECLARE

                ADDR$0         LITERALLY    '0FFC6H',
                ADDR$STATUS    LITERALLY    '0FFC4H',
                CLEAR$FF       LITERALLY    '0FFDDH',
                CMD$37         LITERALLY    '0FFDBH',
                COMMAND$MOD    LITERALLY    '0FFC5H',
                COUNT$HI       LITERALLY    '0FFD1H',
                COUNT$LO       LITERALLY    '0FFD1H',
                CPT$REG        LITERALLY    '0FFC5H',
                EOS$REG        LITERALLY    '0FFC7H',
                PORT$IN        LITERALLY    '0FFC0H',
                PORT$OUT       LITERALLY    '0FFC0H',
                SER$DATA       LITERALLY    '0FFF0H',
                SER$STAT       LITERALLY    '0FFF2H',
                SET$MASK       LITERALLY    '0FFDFH',
                SET$MODE       LITERALLY    '0FFDBH',
                SPOLL$STAT     LITERALLY    '0FFC3H',
                START$HI       LITERALLY    '0FFD0H',
                START$LO       LITERALLY    '0FFD0H',
                STATUS$1       LITERALLY    '0FFC1H',
                STATUS$2       LITERALLY    '0FFC2H';

                /* crt  messages list */

  4    1        DECLARE GET$MSG(11) BYTE DATA (0DH,0AH,'TRIGGER',0AH,0DH);
  5    1        DECLARE DEC$MSG(16) BYTE DATA (0DH,0AH,'DEVICE CLEAR',0AH,0DH);
  6    1        DECLARE REMC$MSG(10) BYTE DATA (0DH,0AH,'REMOTE',0DH,0AH);
  7    1        DECLARE CPT$MSG(22) BYTE DATA (0DH,0AH,'UNDEF CMD RECEIVED',0AH,0DH);
  8    1        DECLARE HUH$MSG(11) BYTE DATA (0DH,0AH,'HUH ???',0DH,0AH);

                    /* called procedures */

  9    1        REGSER:        PROCEDURE;
```

230832-17

```
PL/M-86 COMPILER     HPIB

 10    2              OUTPUT (SPOLL$STAT)=TRG;

 11    2              DO WHILE (INPUT (SPOLL$STAT) AND SRQS)=SRQS;
 12    3              END;

 13    2              OUTPUT (SPOLL$STAT)=NO$RSV;

 14    2              END REQSER;

 15    1       CO: PROCEDURE(XXX);
 16    2              DECLARE
                       XXX          BYTE;

 17    2              DO WHILE (INPUT (SER$STAT) AND TXRDY)<>TXRDY;
 18    3              END;
 19    2              OUTPUT (SER$DATA)=XXX;
 20    2          END CO;
 21    1       HUH:         PROCEDURE;
 22    2                    DO I=0 TO 10;
 23    3                     CALL CO (HUH$MSG(I));
 24    3                    END;
 25    2              END HUH;

 26    1       CI:        PROCEDURE;

 27    2              IF (INPUT (SER$STAT) AND RXRDY)=RXRDY THEN
 28    2                  DO;
 29    3                      I=0;
 30    3                      CHAR$COUNT=0;
 31    3       STORE$CHAR:            CHAR=(INPUT (SER$DATA) AND 7FH);
 32    3                      CHAR$COUNT=CHAR$COUNT+1;
 33    3                      CALL CO (CHAR);
 34    3                      CHARS(I)=CHAR;
 35    3                      I=I+1;
 36    3                      IF CHAR <> CRLF THEN
 37    3                          DO;
 38    4                           DO WHILE (INPUT (SER$STAT) AND RXRDY) <>RXRDY;
 39    5                           END;
 40    4                           GOTO STORE$CHAR;
 41    4                          END;
 42    3                      CALL REQSER;
 43    3                  END;
 44    2              END CI;

 45    1       TALK$EXEC:  PROCEDURE;

 46    2              OUTPUT (STATUS$2)=CLEAR;
                      /*
                      manipulate address bits for DMA controller
                      */

 47    2              DMA$ADR$TALK=(@CHARS);
 48    2              DMA$WRD$TALK(1)=SHL(DMA$WRD$TALK(1),4);
 49    2              DMA$WRD$TALK(0)=DMA$WRD$TALK(0)+DMA$WRD$TALK(1);

 50    2              OUTPUT (CLEAR$FF)=CLEAR;
```

230832-18

```
PL/M-86 COMPILER    HPIB

 51   2          OUTPUT (CMD37)=NORM$TIME;
 52   2          OUTPUT (SET$MODE)=RD$XFER;
 53   2          OUTPUT (SET$MASK)=CLEAR;
 54   2          OUTPUT (START$LO)=DMA$WRD$TALK(0);
 55   2          DMA$WRD$TALK(0)=SHR(DMA$WRD$TALK(0),8);
 56   2          OUTPUT (START$HI)=DMA$WRD$TALK(0);
 57   2          OUTPUT (COUNT$LO)=CHAR$COUNT;
 58   2          OUTPUT (COUNT$HI)=0;


 59   2          OUTPUT (EOS$REG)=EOS;
 60   2          OUTPUT (COMMAND$MOD)=END$EOI;

 61   2          DO WHILE (INPUT (STATUS$1) AND BO)=0;
 62   3          END;
 63   2          OUTPUT (PORT$OUT)=0AAH;

 64   2          DO WHILE (INPUT (STATUS$1) AND ERR)=ERR;
 65   3              DO WHILE (INPUT (STATUS$1) AND BO)=0;
 66   4              END;
 67   3              OUTPUT (PORT$OUT)=0AAH;
 68   3          END;
 69   2          OUTPUT (STATUS$2)=DMA$REG$T;

 70   2          END TALK$EXEC;

 71   1      LISTEN$EXEC:    PROCEDURE;

 72   2          OUTPUT (STATUS$2)=CLEAR;
 73   2          OUTPUT (CLEAR$FF)=CLEAR;
 74   2          OUTPUT (CMD$37)=NORM$TIME;
 75   2          OUTPUT (SET$MODE)=WR$XFER;
 76   2          OUTPUT (SET$MASK)=CLEAR;
 77   2          DMA$ADR$LSTN=(@CHARS);
 78   2          DMA$WRD$LSTN(1)=SHL(DMA$WRD$LSTN(1),4);
 79   2          DMA$WRD$LSTN(0)=DMA$WRD$LSTN(0)+DMA$WRD$LSTN(1);
 80   2          OUTPUT (START$LO)=DMA$WRD$LSTN(0);
 81   2          DMA$WRD$LSTN(0)=SHR(DMA$WRD$LSTN(0),8);
 82   2          OUTPUT (START$HI)=DMA$WRD$LSTN(0);
 83   2          OUTPUT (COUNT$LO)=TC$LO;
 84   2          OUTPUT (COUNT$HI)=TC$HI;
 85   2          OUTPUT (STATUS$2)=DMA$REG$L;

 86   2          END LISTEN$EXEC;

 87   1      PRINTER:    PROCEDURE;

 88   2          I=0;

 89   2          DO WHILE PRI$BUF(I) <>CRLF;
 90   3              CALL CO (PRI$BUF(I));
 91   3              I=I+1;
 92   3          END;
 93   2          CALL CO (PRI$BUF(I));

 94   2          END PRINTER;
```

230832–19

```
PL/M-86 COMPILER    HPIB


     1      ADSC$EXEC:   PROCEDURE;

     2              TA$OR$LA=INPUT (ADDR$STATUS);

     2              IF (TA$OR$LA AND TALK)=TALK THEN
     2                   CALL TALK$EXEC;
     2              IF (TA$OR$LA AND LISTEN)=LISTEN THEN
     2                   CALL LISTEN$EXEC;

     2              END ADSC$EXEC;

     1      GET$EXEC:    PROCEDURE;
     2                   DO I=0 TO 10;
     3                    CALL CO (GET$MSG(I));
     3                   END;
     2              END GET$EXEC;

     1      DEC$EXEC:    PROCEDURE;
     2                   DO I=0 TO 15;
     3                    CALL CO (DEC$MSG(I));
     3                   END;
     2              END DEC$EXEC;

     1      REMC$EXEC:   PROCEDURE;
     2                   DO I=0 TO 9;
     3                    CALL CO (REMC$MSG(I));
     3                   END;
     2              END REMC$EXEC;


     1      PPOLL$CON:   PROCEDURE;

     2              OUTPUT (COMMAND$MOD)=PPOLL$CNFG$FLAG;

     2              END PPOLL$CON;

     1      PPOLL$EN:    PROCEDURE;

     2              PPOLL$EN$BYTE=(UDC AND 6FH);
     2              OUTPUT (COMMAND$MOD)=PPOLL$EN$BYTE;

     2              END PPOLL$EN;

     1      CPT$EXEC:    PROCEDURE;
     2                   DO I=0 TO 21;
     3                    CALL CO (CPT$MSG(I));
     3                   END;

     2                   UDC=INPUT (CPT$REG);
     2                   UDC=(UDC AND 7FH);
     2                   IF (UDC AND PPC)=PPC THEN
     2                    CALL PPOLL$CON;

     2                   IF (UDC AND PPE$MASK)=PPE$MASK THEN
     2                    CALL PPOLL$EN;
```

230832-20

```
PL/M-86 COMPILER    HPIB


 134   2            END CPT$EXEC;
               /*
               BEGIN CODE
               */
 135   1       INIT:

                   OUTPUT  (CLEAR$FF)   =CLEAR;
 136   1           OUTPUT  (COMMAND$MOD)    =RESET;
 137   1           OUTPUT  (ADDR$STATUS)    =MODE$1;
 138   1           OUTPUT  (ADDR$0)     =MLA;
 139   1           OUTPUT  (STATUS$2)   =NO$DMA;
 140   1           OUTPUT  (COMMAND$MOD)    =PON;

 141   1       LISTENERS:

                   /* response to listeners check */


                   DO WHILE (INPUT (STATUS$1) AND BI)=0;
 142   2           END;

 143   1           XYZ=INPUT (PORT$IN);
 144   1           XYZ=INPUT (STATUS$2);

 145   1       CMD:

               RDSTAT:
                   /* read status registers and interpret command */

                   STAT1=INPUT (STATUS$1);
 146   1           STAT2=INPUT (STATUS$2);

 147   1           IF (STAT1 AND DNE)=DNE THEN
 148   1              CALL PRINTER;
 149   1           IF (STAT1 AND CPT)=CPT THEN
 150   1               DO;
 151   2               CALL CPT$EXEC;
 152   2               STAT2=(STAT2 AND 0FEH);
 153   2               END;
 154   1           IF (STAT1 AND GET)=GET THEN
 155   1               DO;
 156   2               CALL GET$EXEC;
 157   2               STAT2=(STAT2 AND 0FEH);
 158   2               END;
 159   1           IF (STAT1 AND DEC)=DEC THEN
 160   1               DO;
 161   2               CALL DEC$EXEC;
 162   2               STAT2=(STAT2 AND 0FEH);
 163   2               END;
 164   1           IF (STAT2 AND REMC)=REMC THEN
 165   1               DO;
 166   2               CALL REMC$EXEC;
 167   2               STAT2=(STAT2 AND 0FEH);
 168   2               END;
 169   1           IF (STAT2 AND ADSC)=ADSC THEN
```

230832-21

```
PL/M-86 COMPILER    HPIB

 170    1                DO;
 171    2                CALL ADSC$EXEC;
 172    2                STAT2=(STAT2 AND OFEH);
 173    2                END;

 174    1            CALL CI;

 175    1            GOTO CMD;


 176    1        END;


MODULE INFORMATION:

     CODE AREA SIZE     = 0475H   1141D
     CONSTANT AREA SIZE = 0000H     0D
     VARIABLE AREA SIZE = 0061H    97D
     MAXIMUM STACK SIZE = 000AH    10D
     349 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

230832-22

APPLICATION NOTE

AP-66

November 1986

Using the 8292 GPIB Controller

Order Number: 231324-002

INTRODUCTION

The Intel® 8292 is a preprogrammed UPI™-41A that implements the Controller function of the IEEE Std 488-1978 (GPIB, HP-IB, IEC Bus, etc.). In order to function the 8292 must be used with the 8291 Talker/Listener and suitable interface and transceiver logic such as a pair of Intel 8293s. In this configuration the system has the potential to be a complete GPIB Controller when driven by the appropriate software. It has the following capabilities: System Controller, send IFC and Take Charge, send REN, Respond to SRQ, send Interface messages, Receive Control, Pass Control, Parallel Poll and Take Control Synchronously.

This application note will explain the 8292 only in the system context of an 8292, 8291, two 8293s and the driver software. If the reader wishes to learn more about the UPI-41A aspects of the 8292, Intel's Application Note AP-41 describes the hardware features and programming characteristics of the device. Additional information on the 8291 may be obtained in the data sheet. The 2893 is detailed in its data sheet. Both chips will be covered here in the details that relate to the GPIB controller.

The next section of this application note presents an overview of the GPIB in a tutorial, but comprehensive nature. The knowledgeable reader may wish to skip this section; however, certain basic semantic concepts introduced there will be used throughout this note.

Additional sections cover the view of the 8292 from the CPU's data bus, the interaction of the 3 chip types (8291, 8292, 8293), the 8292's software protocol and the system level hardware/software protocol. A brief description of interrupts and DMA will be followed by an application example. Appendix A contains the source code for the system driver software.

GPIB/IEEE 488 OVERVIEW

Design Objectives

WHAT IS THE IEEE 488 (GPIB)?

The experience of designing systems for a variety of applications in the early 1970's caused Hewlett-Packard to define a standard intercommunication mechanism which would allow them to easily assemble instrumentation systems of varying degrees of complexity. In a typical situation each instrument designer designed his/her own interface from scratch. Each one was inconsistent in terms of electrical levels, pin-outs on a connector, and types of connectors. Every time they built a system they had to invent new cables and new documentation just to specify the cabling and interconnection procedures.

Based on this experience, Hewlett-Packard began to define a new interconnection scheme. They went further than that, however, for they wanted to specify the typical communication protocol for systems of instruments. So in 1972, Hewlett-Packard came out with the first version of the bus which since has been modified and standardized by a committee of several manufacturers, coordinated through the IEEE, to perfect what is now known as the IEEE 488 Interface Bus (also known as the HPIB, the GPIB and the IEC bus). While this bus specification may not be perfect, it is a good compromise of the various desires and goals of instrumentation and computer peripheral manufacturers to produce a common interconnection mechanism. It fits most instrumentation systems in use today and also fits very well the microcomputer I/O bus requirements. The basic design objectives for the GPIB were to:

1) Specify a system that is easy to use, but has all of the terminology and the definitions related to that system precisely spelled out so that everyone uses the same language when discussing the GPIB.
2) Define all of the mechanical, electrical, and functional interface requirements of a system, yet not define any of the device aspects (they are left up to the instrument designer).
3) Permit a wide range of capabilities of instruments and computer peripherals to use a system simultaneously and not degrade each other's performance.
4) Allow different manufacturers' equipment to be connected together and work together on the same bus.
5) Define a system that is good for limited distance interconnections.
6) Define a system with minimum restrictions on performance of the devices.
7) Define a bus that allows asynchronous communication with a wide range of data rates.
8) Define a low cost system that does not require extensive and elaborate interface logic for the low cost instruments, yet provides higher capability for the higher cost instruments if desired.
9) Allow systems to exist that do not need a central controller; that is, communication directly from one instrument to another is possible.

Although the GPIB was originally designed for instrumentation systems, it became obvious that most of these systems would be controlled by a calculator or computer. With this in mind several modifications were made to the original proposal before its final adoption as an international standard. Figure 1 lists the salient characteristics of the GPIB as both an instrumentation bus and as a computer I/O bus.

Data Rate
- 1M bytes/s, max
- 250k bytes/s, typ

Multiple Devices
- 15 devices, max (electrical limit)
- 8 devices, typ (interrupt flexibility)

Bus Length
- 20 m, max
- 2 m/device, typ

Byte Oriented
- 8-bit commands
- 8-bit data

Block Multiplexed
- Optimum strategy on GPIB due to setup overhead for commands

Interrupt Driven
- Serial poll (slower devices)
- Parallel poll (faster devices)

Direct Memory Access
- One DMA facility at controller serves all devices on bus

Asynchronous
- One talker } 3-wire handshake
- Multiple listeners }

I/O to I/O Transfers
- Talker and listeners need not include microcomputer/controller

Figure 1. Major Characteristics of GPIB as Microcomputer I/O Bus

The bus can be best understood by examining each of these characteristics from the viewpoint of a general microcomputer I/O bus.

Data Rate—Most microcomputer systems utilize peripherals of differing operational rates, such as floppy discs at 31k or 62k bytes/s (single or double density), tape cassettes at 5k to 10k bytes/s, and cartridge tapes at 40k to 80k bytes/s. In general, the only devices that need high speed I/O are 0.5″ (1.3-cm) magnetic tapes and hard discs, operational at 30k to 781k bytes/s, respectively. Certainly, the 250k-bytes/s data rate that can be easily achieved by the IEEE 488 bus is sufficient for microcomputers and their peripherals, and is more than needed for typical analog instruments that take only a few readings per second. The 1M-byte/s maximum data rate is not easily achieved on the GPIB and requires special attention to considerations beyond the scope of this note. Although not required, data buffering in each device will improve the overall bus performance and allow utilization of more of the bus bandwidth.

Multiple Devices—Many microcomputer systems used as computers (not as components) service from three to seven peripherals. With the GPIB, up to 8 devices can be handled easily by 1 controller; with some slowdown in interrupt handling, up to 15 devices can work together. The limit of 8 is imposed by the number of unique parallel poll responses available; the limit of 15 is set by the electrical drive characteristics of the bus. Logically, the IEEE 488 Standard is capable of accommodating more device addresses (31 primary, each potentially with 31 secondaries).

Bus Length—Physically, the majority of microcomputer systems fit easily on a desk top or in a standard 19″ (48-cm) rack, eliminating the need for extra long cables. The GPIB is designed typically to have 2m of length per device, which accommodates most systems. A line printer might require greater cable lengths, but this can be handled at the lower speeds involved by using extra dummy terminations.

Byte Oriented—The 8-bit byte is almost universal in I/O applications; even 16-bit and 32-bit computers use byte transfers for most peripherals. The 8-bit byte matches the ASCII code for characters and is an integral submultiple of most computer word sizes. The GPIB has an 8-bit wide data path that may be used to transfer ASCII or binary data, as well as the necessary status and control bytes.

Block Multiplexed—Many peripherals are block oriented or are used in a block mode. Bytes are transferred in a fixed or variable length group; then there is a wait before another group is sent to that device, e.g., one sector of a floppy disc, one line on a printer or type punch, etc. The GPIB is, by nature, a block multiplexed bus due to the overhead involved in addressing various devices to talk and listen. This overhead is less bothersome if it only occurs once for a large number of data bytes (once per block). This mode of operation matches the needs of microcomputers and most of their peripherals. Because of block multiplexing, the bus works best with buffered memory devices.

Interrupt Driven—Many types of interrupt systems exist, ranging from complex, fast, vectored/priority networks to simple polling schemes. The main tradeoff is usually cost versus speed of response. The GPIB has two interrupt protocols to help span the range of applications. The first is a single service request (SRQ) line that may be asserted by all interrupting devices. The controller then polls all devices to find out which wants service. The polling mechanism is well defined and can

be easily automated. For higher performance, the parallel poll capability in the IEEE 488 allows up to eight devices to be polled at once—each device is assigned to one bit of the data bus. This mechanism provides fast recognition of an interrupting device. A drawback is the frequent need for the controller to explicitly conduct a parallel poll, since there is no equivalent of the SRQ line for this mode.

Direct Memory Access (DMA)—In many applications, no immediate processing of I/O data on a byte-by-byte basis is needed or wanted. In fact, programmed transfers slow down the data transfer rate unnecessarily in these cases, and higher speed can be obtained using DMA. With the GPIB, one DMA facility at the controller serves all devices. There is no need to incorporate complex logic in each device.

Asynchronous Transfers—An asynchronous bus is desirable so that each device can transfer at its own rate. However, there is still a strong motivation to buffer the data at each device when used in large systems in order to speed up the aggregate data rate on the bus by allowing each device to transfer at top speed. The GPIB is asynchronous and uses a special 3-wire handshake that allows data transfers from one talker to many listeners.

I/O to I/O Transfers—In practice, I/O to I/O transfers are seldom done due to the need for processing data and changing formats or due to mismatched data rates. However, the GPIB can support this mode of operation where the microcomputer is neither the talker nor one of the listeners.

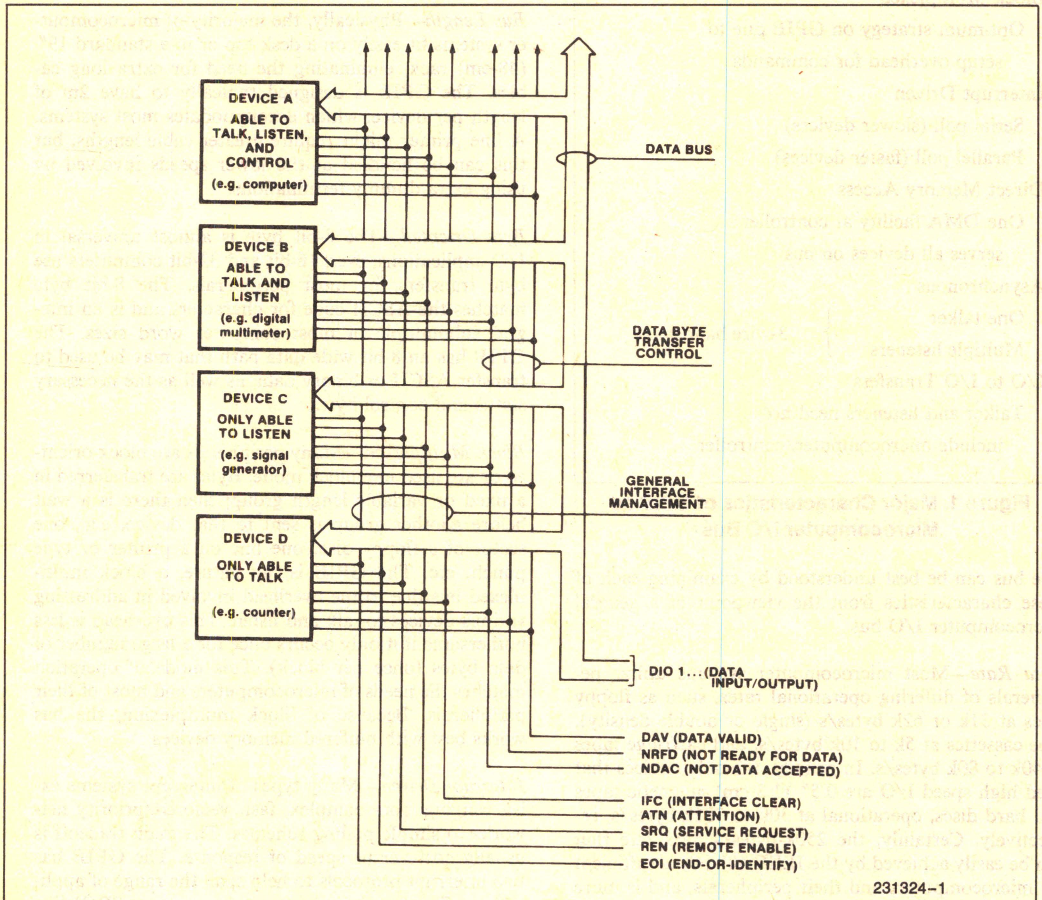


Figure 2. Interface Capabilities and Bus Structure

GPIB Signal Lines

DATA BUS

The lines DI01 through DI08 are used to transfer addresses, control information and data. The formats for addresses and control bytes are defined by the IEEE 488 standard (see Appendix C). Data formats are undefined and may be ASCII (with or without parity) or binary. D101 is the Least Significant bit (note that this will correspond to bit 0 on most computers).

MANAGEMENT BUS

ATN—Attention. This signal is asserted by the Controller to indicate that it is placing an address or control byte on the Data Bus. ATN is de-asserted to allow the assigned Talker to place status or data on the Data Bus. The Controller regains control by reasserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.

EOI—End or Identify. This signal has two uses as its name implies. A talker may assert EOI simultaneously with the last byte of data to indicate end of data. The Controller may assert EOI along with ATN to initiate a Parallel Poll. Although many devices do not use Parallel Poll, all devices *should* use EOI to end transfers (many currently available ones do not).

SRQ—Service Request. This line is like an interrupt: it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a Serial Poll at its earliest convenience. The device deasserts SRQ when polled.

IFC—Interface Clear. This signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After deasserting IFC, the System Controller is the active controller of the system.

REN—Remote Enable. This signal is asserted only by the System Controller. Its assertion does not place devices into Remote Control mode; REN only *enables* a device to go remote when addressed to listen. When in Remote, a device should ignore its front panel controls.

TRANSFER BUS

NRFD—Not Ready For Data. This handshake line is asserted by a listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD deasserted (i.e., ready for data) until all devices have deasserted NRFD.

NDAC—Not Data Accepted. This handshake line is asserted by a Listener to indicate it has not yet accepted the data or control byte on the DIO lines. Note that the Controller will not see NDAC deasserted (i.e., data accepted) until all devices have deasserted NDAC.

DAV—Data Valid. This handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the DIO lines and has had the minimum specified settling time.

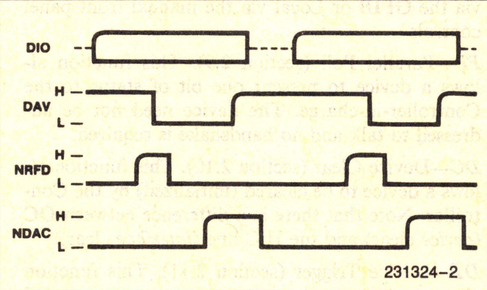


Figure 3. GPIB Handshake Sequence

GPIB Interface Functions

There are ten (10) interface functions specified by the IEEE 488 standard. Not all devices will have all functions and some may only have partial subsets. The ten functions are summarized below with the relevant section number from the IEEE document given at the beginning of each paragraph. For further information please see the IEEE standard.

1) *SH*—Source Handshake (section 2.3). This function provides a device with the ability to properly transfer data from a Talker to one or more Listeners using the three handshake lines.
2) *AH*—Acceptor Handshake (section 2.4). This function provides a device with the ability to properly receive data from the Talker using the three handshake lines. The AH function may also delay the beginning (NRFD) or end (NDAC) of any transfer.
3) *T*—Talker (section 2.5). This function allows a device to send status and data bytes when addressed to talk. An address consists of one (Primary) or two (Primary and Secondary) bytes. The latter is called an extended Talker.

4) *L*—Listener (section 2.6). This function allows a device to receive data when addressed to listen. There can be extended Listeners (analogous to extended Talkers above).

5) *SR*—Service Request (section 2.7). This function allows a device to request service (interrupt) the Controller. The SRQ line may be asserted asynchronously.

6) *RL*—Remote Local (section 2.8). This function allows a device to be operated in two modes: Remote via the GPIB or Local via the manual front panel controls.

7) *PP*—Parallel Poll (section 2.9). This function allows a device to present one bit of status to the Controller-in-charge. The device need not be addressed to talk and no handshake is required.

8) *DC*—Device Clear (section 2.10). This function allows a device to be cleared (initialized) by the Controller. Note that there is a difference between DC (*device* clear) and the IFC line (*interface* clear).

9) *DT*—Device Trigger (section 2.11). This function allows a device to have its basic operation started either individually or as part of a group. This capability is often used to synchronize several instruments.

10) *C*—Controller (section 2.12). This function allows a device to send addresses, as well as universal and addressed commands to other devices. There may be more than one controller on a system, but only one may be the controller-in-charge at any one time.

At power-on time the controller that is hardwired to be the System Controller becomes the active controller-in-charge. The System Controller has several unique capabilities including the ability to send Interface Clear (IFC—clears all device interfaces and returns control to the System Controller) and to send Remote Enable (REN—allows devices to respond to bus data once they are addressed to listen). The System Controller may optionally Pass Control to another controller, if the system software has the capability to do so.

GPIB Connector

The GPIB connector is a standard 24-pin industrial connector such as Cinch or Amphenol series 57 Micro-Ribbon. The IEEE standard specifies this connector, as well as the signal connections and the mounting hardware.

The cable has 16 signal lines and 8 ground lines. The maximum length is 20 meters with no more than two meters per device.

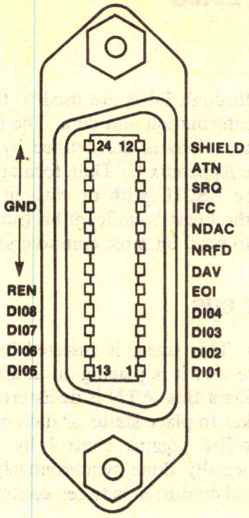


Figure 4. GPIB Connector

GPIB Signal Levels

The GPIB signals are all TTL compatible, low true signals. A signal is asserted (true) when its electrical voltage is less than 0.5 volts and is deasserted (false) when it is greater than 2.4 volts. Be careful not to become confused with the two handshake signals, NRFD and NDAC which are also low true (i.e. > 0.5 volts implies the device is Not Ready For Data).

The Intel 8293 GPIB transceiver chips ensure that all relevant bus driver/receiver specifications are met. Detailed bus electrical specifications may be found in Section 3 of the IEEE Std 488-1978. The Standard is the ultimate reference for all GPIB questions.

GPIB Message Protocols

The GPIB is a very flexible communications medium and as such has many possible variations of protocols. To bring some order to the situation, this section will discuss a protocol similar to the one used by Ziatech's ZT80 GPIB controller for Intel's MULTIBUS™ computers. The ZT80 is a complete high-level interface processor that executes a set of high level instructions that map directly into GPIB actions. The sequences of commands, addresses and data for these instructions provide a good example of how to use the GPIB (additional information is available in the ZT80 Instruction Manual). The 'null' at the end of each instruction is for cosmetic use to remove previous information from the DIO lines.

DATA—Transfer a block of data from device A to devices B, C . . .

1) Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2) Universal Unlisten
3) Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   Device C Primary (Listen) Address
   etc.
4) First Data Byte
   Second Data Byte
   .
   .
   .
   Last Data Byte (EOI)
5) Null

TRIGR—Trigger devices A, B . . . to take action

1) Universal Unlisten
2) Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3) Group Execute Trigger
4) Null

PSCTL—Pass control to device A

1) Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2) Talk Control
3) Null

CLEAR—Clear all devices

1) Device Clear
2) Null

REMAL—Remote Enable

1) Assert REN continuously

GOREM—Put devices A, B, . . . into Remote

1) Assert REN continuously
2) Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3) Null

GOLOC—Put devices A, B, . . . into Local

1) Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
2) Go To Local
3) Null

LOCAL—Reset all devices to Local

1) Stop asserting REN

LLKAL—Prevent all devices from returning to Local

1) Local Lock Out
2) Null

SPOLL—Conducts a serial poll of devices A, B, . . .

1) Serial Poll Enable
2) Universal Unlisten
3) ZT 80 Primary (Listen) Address
   ZT 80 Secondary Address
4) Device Primary (Talk) Address
   Device Secondary Address (if any)
5) Status byte from device
6) Go to Step 4 until all devices on list have been polled
7) Serial Poll Disable
8) Null

PPUAL—Unconfigure and disable Parallel Poll response from all devices

1) Parallel Poll Unconfigure
2) Null

ENAPP—Enable Parallel Poll response in devices A, B, . . .

1) Universal Unlisten
2) Device Primary (Listen) Address
   Device Secondary Address (if any)
3) Parallel Poll Configure
4) Parallel Poll Enable
5) Go to Step 2 until all devices on list have been configured.
6) Null

DISPP—Disable Parallel Poll response from devices A, B, . . .

1) Universal Unlisten
2) Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3) Disable Parallel Poll
4) Null

This Ap Note will detail how to implement a useful subset of these controller instructions.

HARDWARE ASPECTS OF THE SYSTEM

8291 GPIB Talker/Listener

The 8291 is a custom designed chip that implements many of the non-controller GPIB functions. It provides hooks so the user's software can implement additional features to complete the set. This chip is discussed in detail in its data sheet. The major features are summarized here:

- Designed to interface microprocessors to the GPIB
- Complete Source and Acceptor Handshake
- Complete Talker and Listener Functions with extended addressing
- Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local functions
- Programmable data transfer rate
- Maskable interrupts
- On-chip primary and secondary address recognition
- 1–8 MHz clock range
- 16 registers (8 read, 8 write) for CPU interface
- DMA handshake provision
- Trigger output pin
- On-chip EOS (End of Sequence)

The pinouts and block diagram are shown in Figure 5. One of eight read registers is for data transfer to the CPU; the other seven allow the microprocessor to monitor the GPIB states and various bus and device conditions. One of the eight write registers is for data transfer from the CPU; the other seven control various features of the 8291.

The 8291 interface functions will be software configured in this application example to the following subsets for use with the 8292 as a controller that does not pass control. The 8291 is used only to provide the handshake logic and to send and receive data bytes. It is not acting as a normal device in this mode, as it never sees ATN asserted.

SH1	Source Handshake
AH1	Acceptor Handshake
T3	Basic Talk-Only
L1	Basic Listen-Only
SR0	No Service Requests
RL0	No Remote/Local
PP0	No Parallel Poll Response
DC0	No Device Clear
DT0	No Device Trigger

If control is passed to another controller, the 8291 must be reconfigured to act as a talker/listener with the following subsets:

SH1	Source Handshake
AH1	Acceptor Handshake
T5	Basic Talker and Serial Poll
L3	Basic Listener
SR1	Service Requests
RL1	Remote/Local with Lockout
PP2	Reconfigured Parallel Poll
DC1	Device Clear
DT1	Device Trigger
C0	Not a Controller

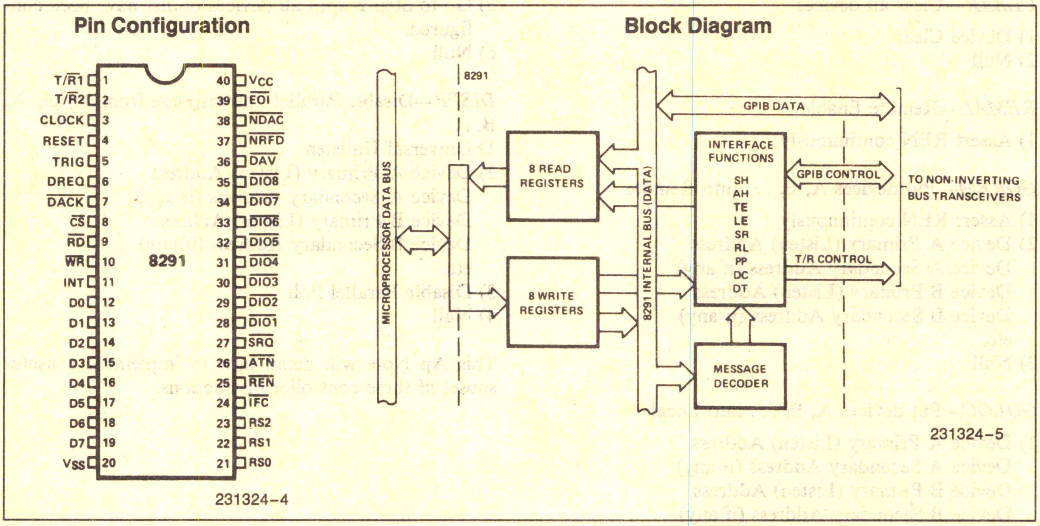


Figure 5. 8291 Pin Configuration and Block Diagram

Most applications do not pass control and the controller is always the system controller (see 8292 commands below).

8292 GPIB Controller

The 8292 is a preprogrammed Intel 8051A that provides the additional functions necessary to implement a GPIB controller when used with an 8291 Talker/Listener. The 8041A is documented in both a user's manual and in AP-41. The following description will serve only as an outline to guide the later discussion.

The 8292 acts as an intelligent slave processor to the main system CPU. It contains a processor, memory, I/O and is programmed to perform a variety of tasks associated with GPIB controller operation. The on-chip RAM is used to store information about the state of the Controller function, as well as a variety of local variables, the stack and certain user status information. The timer/counter may be optionally used for several time-out functions or for counting data bytes transferred. The I/O ports provide the GPIB control signals, as well as the ancillary lines necessary to make the 8291, 2, 3 work together.

The 8292 is closely coupled to the main CPU through three on-chip registers that may be independently accessed by both the master and the 8292 (UPI-41A). Figure 6 shows this Register Interface. Also refer to Figure 12.

The status register is used to pass Interrupt Status information to the master CPU (A0 = 1 on a read).

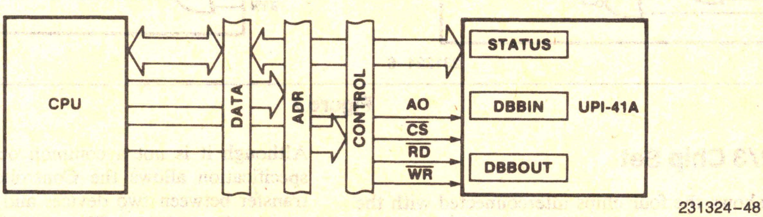
The DBBOUT register is used to pass one of five other status words to the master based on the last command written into DBBIN. DBBOUT is accessed when A0 = 0 on a Read. The five status words are Error Flag, Controller Status, GPIB Status, Event Counter Status or Time Out Status.

DBBIN receives either commands (A0 = 1 on a Write) or command related data (A0 = 0 on a write) from the master. These command related data are Interrupt Mask, Error Mask, Event Counter or Time Out.

8293 GPIB Transceivers

The 8293 is a multi-use HMOS chip that implements the IEEE 488 bus transceivers and contains the additional logic required to make the 8291 and 8292 work together. The two option strapping pins are used to internally configure the chip to perform the specialized gating required for use with 8291 as a device or with 8291/92 as a controller.

In this application example the two configurations used are shown in Figure 7a and 7b. The drivers are set to open collector or three state mode as required and the special logic is enabled as required in the two modes.



$\overline{CS}$	AO	$\overline{RD}$	$\overline{WR}$	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

Figure 6. UPI-41A Registers

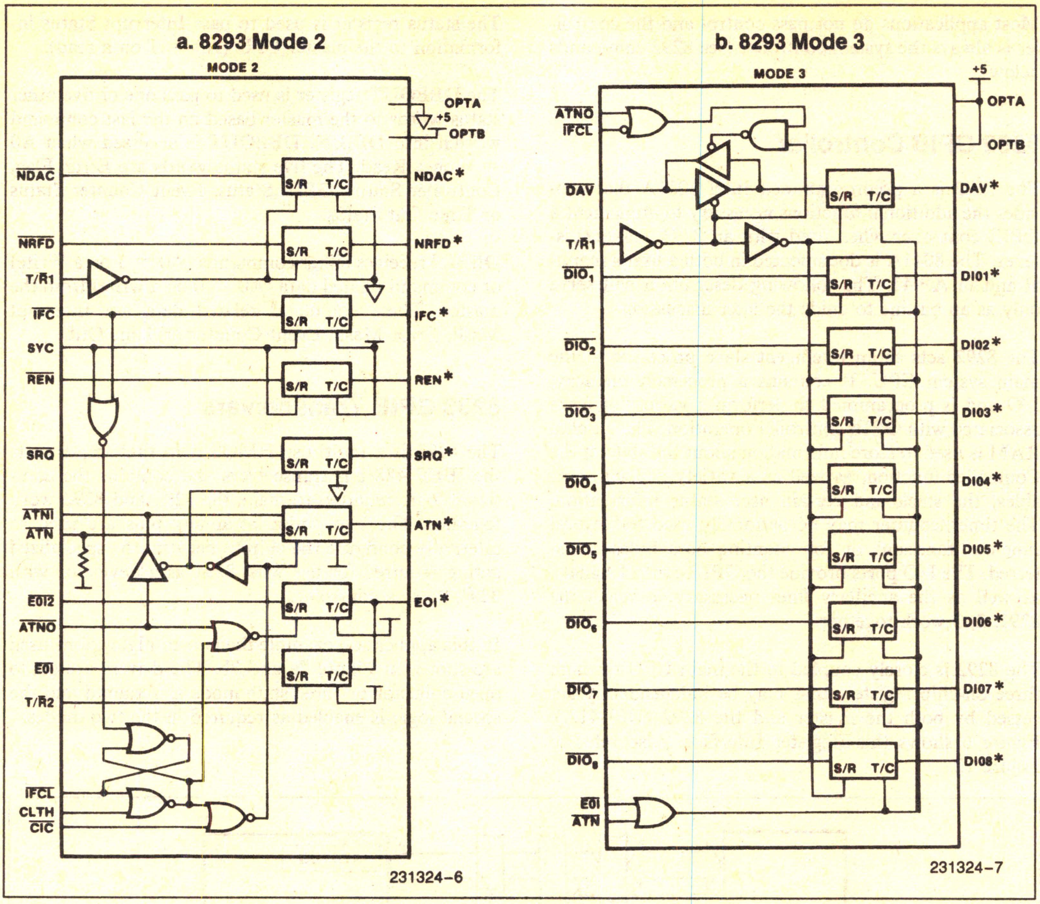


Figure 7

8291/2/3 Chip Set

Figure 8 shows the four chips interconnected with the special logic explicitly shown.

The 8291 acts only as the mechanism to put commands and addresses on the bus while the 8292 is asserting ATN. The 8291 is tricked into believing that the ATN line is not asserted by the ATN2 output of the ATN transceiver and is placed in Talk-only mode by the CPU. The 8291 then acts as though it is sending data, when in reality it is sending addresses and/or commands. When the 8292 deasserts ATN, the CPU software must place the 8291 in Talk-only, Listen-only or Idle based on the implicit knowledge of how the controller is going to participate in the data transfer. In other words, the 8291 does not respond directly to addresses or commands that it sends on the bus on behalf of the Controller. The user software, through the use of Listen-only or Talk-only, makes the 8291 behave as though it were addressed.

Although it is not a common occurrence, the GPIB specification allows the Controller to set up a data transfer between two devices and not directly participate in the exchange. The controller must know when to go active again and regain control. The chip set accomplishes this through use of the "Continuous Acceptor Handshake cycling mode" and the ability to detect EOI or EOS at the end of the transfer. See XFER in the Software Driver Outline below.

If the 8292 is not the System Controller as determined by the signal on its SYC pin, then it must be able to respond to an IFC within 100 μsec. This is accomplished by the cross-coupled NORs in Figure 7a which deassert the 8293's internal version of $\overline{\text{CIC}}$ (Not Controller-in-Charge). This condition is latched until the 8292's firmware has received the IFCL (interface clear received latch) signal by testing the $\overline{\text{IFCL}}$ input. The firmware then sets its signals to reflect the inactive condition and clears the 8293's latch.

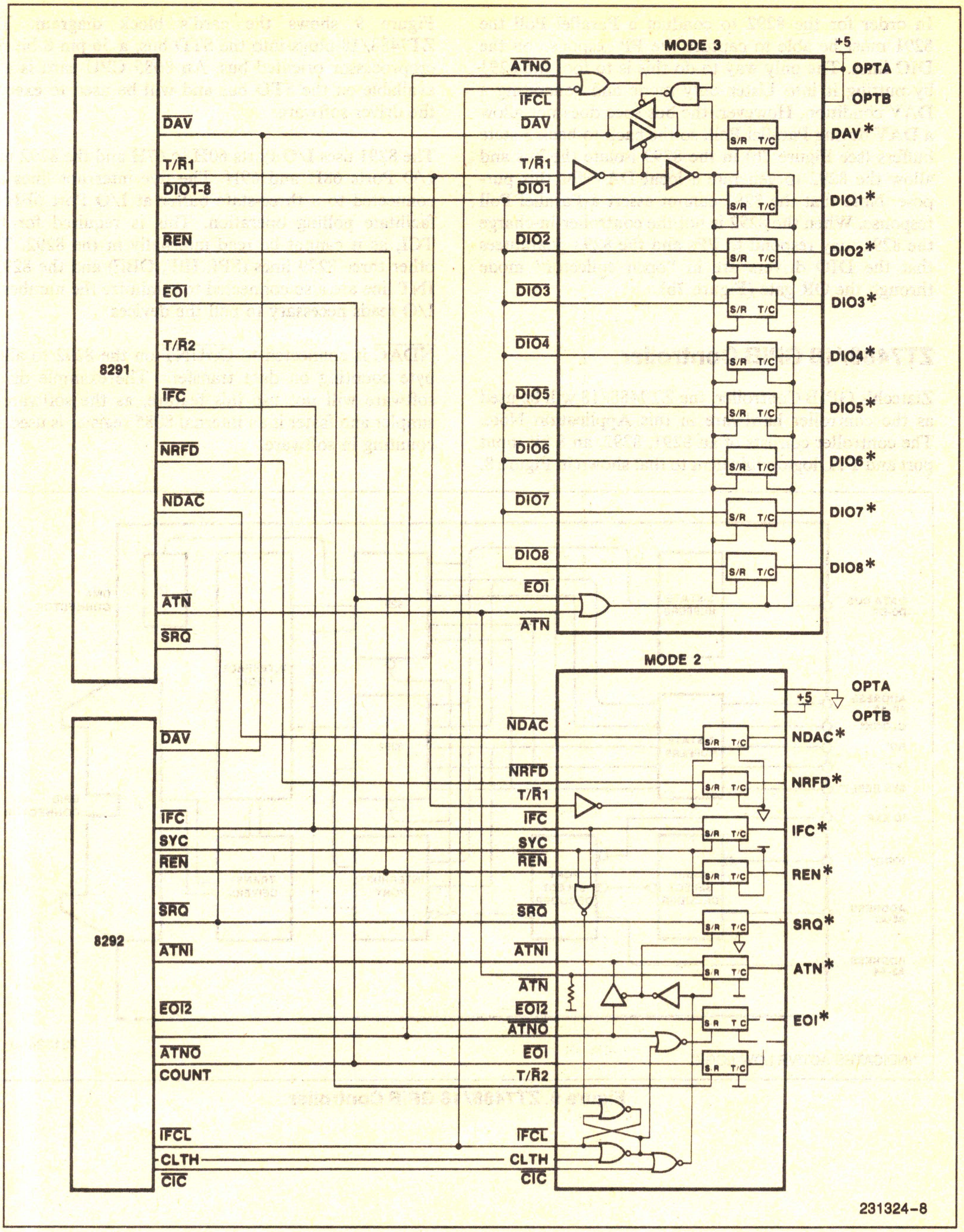


Figure 8. Talker/Listener/Controller

In order for the 8292 to conduct a Parallel Poll the 8291 must be able to capture the PP response on the DIO lines. The only way to do this is to fool the 8291 by putting it into Listen-only mode and generating a DAV condition. However, the bus spec does not allow a DAV during Parallel Poll, so the back-to-back 3-state buffers (see Figure 7b) in the 8293 isolate the bus and allow the 8292 to generate a local DAV for this purpose. Note that the 8291 cannot assert a Parallel Poll response. When the 8292 is not the controller-in-charge the 8291 may respond to PPs and the 8293 guarantees that the DIO drivers are in "open collector" mode through the OR gate (Figure 7b).

ZT7488/18 GPIB Controller

Ziatech's GPIB Controller, the ZT7488/18 will be used as the controller hardware in this Application Note. The controller consists of an 8291, 8292, an 8 bit input port and TTL logic equivalent to that shown in Figure 8.

Figure 9 shows the card's block diagram. The ZT7488/18 plugs into the STD bus, a 56 pin 8 bit microprocessor oriented bus. An 8085 CPU card is also available on the STD bus and will be used to execute the driver software.

The 8291 uses I/O Ports 60H to 67H and the 8292 uses I/O Ports 68H and 69H. The five interrupt lines are connected to a three-state buffer at I/O Port 6FH to facilitate polling operation. This is required for the TCI, as it cannot be read internally in the 8292. The other three 8229 lines (SPI, IBF, OBF) and the 8291's INT line are also connected to minimize the number of I/O reads necessary to poll the devices.

$\overline{\text{NDAC}}$ is connected to $\overline{\text{COUNT}}$ on the 8292 to allow byte counting on data transfers. The example driver software will not use this feature, as the software is simpler and faster if an internal 8085 register is used for counting in software.

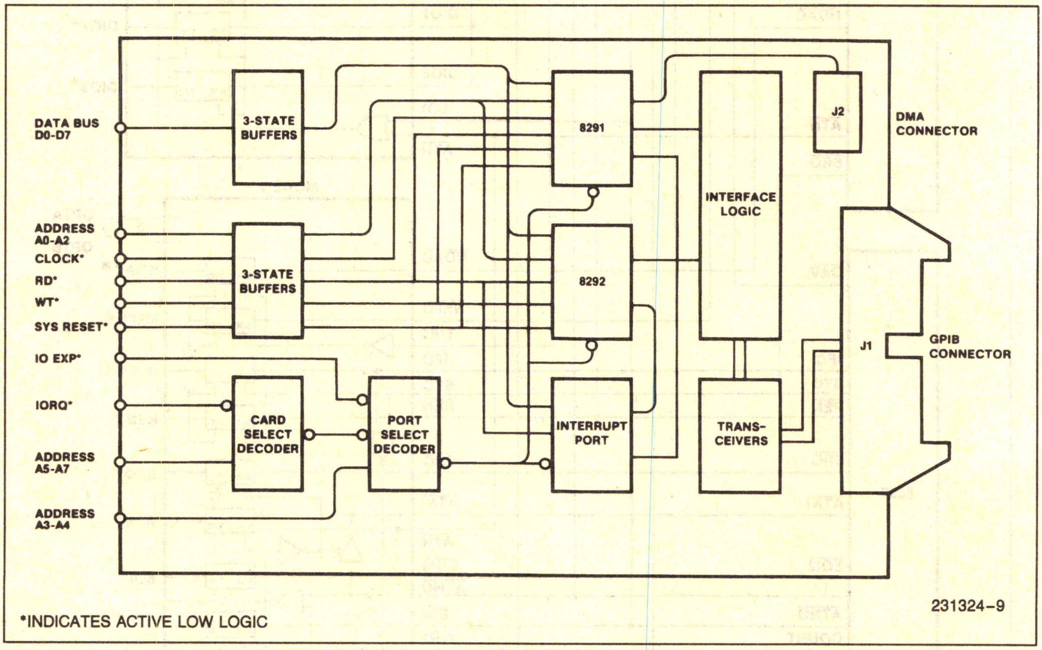


Figure 9. ZT7488/18 GPIB Controller

READ REGISTERS	D7	D6	D5	D4	D3	D2	D1	D0	PORT #	D7	D6	D5	D4	D3	D2	D1	D0	WRITE REGISTERS
DATA IN	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	6φH	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0	DATA OUT
INTERRUPT STATUS 1	CPT	APT	GET	END	DEC	ERR	BO	BI	61H	CPT	APT	GET	END	DEC	ERR	BO	BI	INTERRUPT MASK 1
INTERRUPT STATUS 2	INT	SPAS	LLO	REM	SPASC	LLOC	REMC	ADSC	62H	0	0	DMAO	DMAI	SPASC	LLOC	REMC	ADSC	INTERRUPT MASK 2
SERIAL POLL STATUS	S8	SRQS	S6	S5	S4	S3	S2	S1	63H	S8	rsv	S6	S5	S4	S3	S2	S1	SERIAL POLL MODE
ADDRESS STATUS	ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN	64H	TO	LO	0	0	0	0	ADM1	ADM0	ADDRESS MODE
COMMAND PASS THROUGH	CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	65H	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0	AUX MODE
ADDRESS 0	X	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0	66H	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1	ADDRESS 0/1
ADDRESS 1	X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1	67H	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0	EOS

Figure 10. 8291 Registers

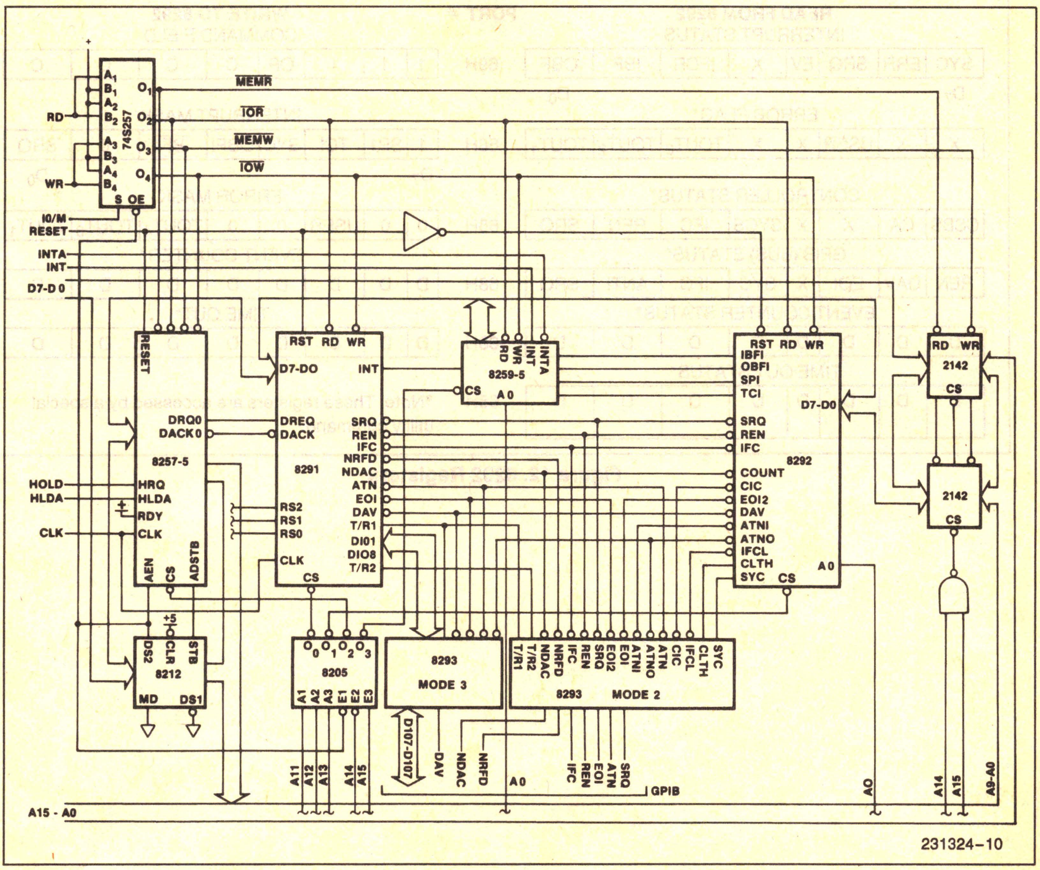


Figure 11. DMA/Interrupt GPIB Controller Block Diagram

The application example will not use DMA or interrupts; however, the Figure 11 block diagram includes these features for completeness.

The 8257-5 DMA chip can be used to transfer data between the RAM and the 8291 Talker/Listener. This mode allows a faster data rate on the GPIB and typically will depend on the 8291's EOS or EOI detection to terminate the transfer. The 8259-5 interrupt controller is used to vector the five possible interrupts for rapid software handling of the various conditions.

8292 COMMAND DESCRIPTION

This section discusses each command in detail and relates them to a particular GPIB activity. Recall that although the 8041A has only two read registers and one write register, through the magic of on-chip firmware the 8292 appears to have six read registers and five write registers. These are listed in Figure 12. Please see the 8292 data sheet for detailed definitions of each register. Note the two letter mnemonics to be used in later discussions. The CPU must not write into the 8292 while IBF (Input Buffer Full) is a one, as information will be lost.

Direct Commands

Both the Interrupt Mask (IM) and the Error Mask (EM) register may be directly written with the LSB of the address bus (A0) a "0". The firmware uses the MSB of the data written to differentiate between IM and EM.

LOAD INTERRUPT MASK

This command loads the Interrupt Mask with D7–D0. Note that D7 must be a "1" and that interrupts are enabled by a corresponding "1" bit in this register. IFC interrupt cannot be masked off; however, when the 8292 is the System Controller, sending an ABORT command will not cause an IFC interrupt.

READ FROM 8292

Register	D_7							D_0	PORT #
INTERRUPT STATUS	SYC	ERR	SRQ	EV	X	IFCR	IBF	OBF	69H
ERROR FLAG*	X	X	USER	X	X	$TOUT_3$	$TOUT_2$	$TOUT_1$	68H
CONTROLLER STATUS*	CSBS	CA	X	X	SYCS	IFC	REN	SRQ	68H
GPIB (BUS) STATUS*	REN	DAV	EOI	X	SYC	IFC	ANTI	SRQ	68H
EVENT COUNTER STATUS*	D	D	D	D	D	D	D	D	68H
TIME OUT STATUS*	D	D	D	D	D	D	D	D	68H

WRITE TO 8292

PORT #	Register	D_7							D_0
69H	COMMAND FIELD	1	1	1	OP	C	C	C	C
68H	INTERRUPT MASK	1	SP1	TCI	SYC	OBFI	$\overline{IBFI}$	0	SRQ
68H	ERROR MASK	0	0	USER	0	0	$TOUT_4$	$TOUT_3$	$TOUT_1$
68H	EVENT COUNTER*	D	D	D	D	D	D	D	D
68H	TIME OUT*	D	D	D	D	D	D	D	D

*Note: These registers are accessed by a special utility command.

Figure 12. 8292 Registers

LOAD ERROR MASK

This command loads the Error Mask with D7–D0. Note that D7 must be a zero and that interrupts are enabled by a corresponding "1" bit in this register.

Utility Commands

These commands are used to read or write the 8292 registers that are not directly accessible. All utility commands are written with A0 = 1, D7 = D6 = D5 = 1, D4 = 0. D3–D0 specify the particular command. For writing into registers the general sequence is:

1) wait for IBF = 0 in Interrupt Status Register
2) write the appropriate command to the 8292,
3) write the desired register value to the 8292 with A0 = 1 with no other writes intervening,
4) wait for indication of completion from 8292 (IBF = 0).

For reading a register the general sequence is:

1) wait for IBF = 0 in Interrupt Status Register
2) write the appropriate command to the 8292
3) wait for a TCI (Task Complete Interrupt)
4) Read the value of the accessed register from the 8292 with A0 = 0.

WEVC—Write to Event Counter
(Command = 0E2H)

The byte written following this command will be loaded into the event counter register and event counter status for byte counting. The internal counter is incremented on a high to low transition of the COUNT (T1) input. In this application example $\overline{\text{NDAC}}$ is connected to count. The counter is an 8 bit register and therefore can count up to 256 bytes (writing 0 to the EC implies a count of 256). If longer blocks are desired, the main CPU must handle the interrupts every 256 counts and carefully observe the timing constraints.

Because the counter has a frequency range from 0 to 133 kHz when using a 6 MHz crystal, this feature may not be usable with all devices on the GPIB. The 8291 can easily transfer data at rates up to 250 kHz and even faster with some tuning of the system. There is also a 500 ns minimum high time requirement for COUNT which can potentially be violated by the 8291 in continuous acceptor handshake mode (i.e., TNDDV1 + TDVND2 − C = 350 + 350 = 700 max). When cable delays are taken into consideration, this problem will probably never occur.

When the 8292 has completed the command, IBF will become a "0" and will cause an interrupt if masked on.

WTOUT—Write to Time Out Register
(Command = 0E1H)

The byte written following this command will be used to determine the number of increments used for the time out functions. Because the register is 8 bits, the maximum time out is 256 time increments. This is probably enough for most instruments on the GPIB but is not enough for a manually stepped operation using a GPIB logic analyzer like Ziatech's ZT488. Also, the 488 Standard does not set a lower limit on how long a device may take to do each action. Therefore, any use of a time out must be able to be overridden (this is a good general design rule for service and debugging considerations).

The time out function is implemented in the 8292's firmware and will not be an accurate time. The counter counts backwards to zero from its initial value. The function may be enabled/disabled by a bit in the Error mask register. When the command is complete IBF will be set to a "0" and will cause an interrupt if masked on.

REVC—Read Event Counter Status
(Command = 0E3H)

This command transfers the content of the Event Counter to the DBBOUT register. The firmware then sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value from the 8292 with A0 = 0.

RINM—Read Interrupt Mask Register
(Command = 0E5H)

This command transfers the content of the Interrupt Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RERM—Read Error Mask Register
(Command = 0EAH)

This command transfers the content of the Error Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RCST—Read Controller Status Register
(Command = 0E6H)

This command transfers the content of the Controller Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RTOUT—Read Time Out Status Register
(Command = 0E9H)

This command transfers the content of the Time Out Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

If this register is read while a time-out function is in process, the value will be the time remaining before time-out occurs. If it is read after a time-out, it will be zero. If it is read when no time-out is in process, it will be the last value reached when the previous timing occurred.

RBST—Read Bus Status Register
(Command = 0E7H)

This command causes the firmware to read the GPIB management lines, DAV and the SYC pin and place a copy in DBBOUT. TCI is set to "1" and will cause an interrupt if masked on. The CPU may read the value.

RERF—Read Error Flag Register
(Command = 0E4H)

This command transfers the content of the Error Flag register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

This register is also placed in DBBOUT by an IACK command if ERR remains set. TCI is set to "1" in this case also.

IACK—Interrupt Acknowledge
(Command = A1 A2 A3 A4 1 A5 1 1)

This command is used to acknowledge any combinations of the five SPI interrupts (A1–A5): SYC, ERR, SRQ, EV, and IFCR. Each bit A1–A5 is an individual acknowledgement to the corresponding bit in the Interrupt Status Register. The command clears SPI but it will be set again if all of the pending interrupts were not acknowledged.

If A2 (ERR) is "1", the Error Flag register is placed in DBBOUT and TCI is set. The CPU may then read the Error Flag without issuing an RERF command.

Operation Commands

The following diagram (Figure 13) is an attempt to show the interrelationships among the various 8292 Operation Commands. It is not meant to replace the complete controller state diagram in the IEEE Standard.

RST—Reset (Command = 0F2H)

This command has the same effect as an external reset applied to the chip's pin #4. The 8292's actions are:

1) All outputs go to their electrical high state. This means that SPI, TCI, OBFI, $\overline{IBFI}$, CLTH will be TRUE and all other GPIB signals will be FALSE.
2) The 8292's firmware will cause the above mentioned five signals to go FALSE after approximately 17.5 μs (at 6 MHz).
3) These registers will be cleared: Interrupt Status, Interrupt Mask, Error Mask, Time Out, Event Counter, Error Flag.
4) If the 8292 is the System Controller (SYC is TRUE), then IFC will be sent TRUE for approximately 100 μs and the Controller function will end up in charge of the bus. If the 8292 is not the System Controller then it will end up in an Idle state.
5) TCI will not be set.

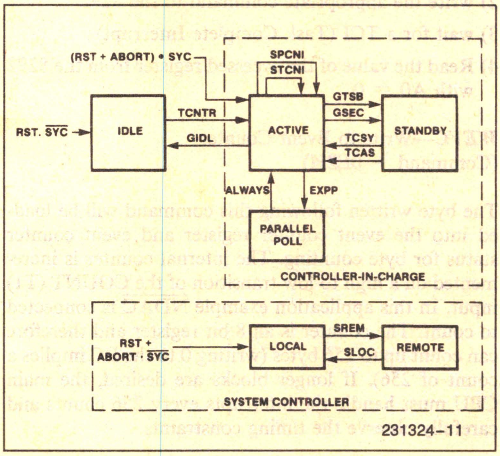


Figure 13. 8292 Command Flowchart

RSTI—Reset Interrupts (Command = 0F3)

This command clears all pending interrupts and error flags. The 8292 will stop waiting for actions to occur (e.g., waiting for ATN to go FALSE in a TCNTR command or waiting for the proper handshake state in a TCSY command). TCI will not be set.

ABORT—Abort all operations and Clear Interface
(Command = 0F9H)

If the 8292 is not the System Controller this command acts like a NOP and flags a USER ERROR in the Error Flag Register. No TCI will occur.

If the 8292 is the system Controller then IFC is set TRUE for approximately 100 μs and the 8292 becomes the Controller-in-Charge and asserts ATN. TCI will be set, only if the 8292 was NOT the CIC.

STCNI—Start Counter Interrupts
(Command = 0FEH)

Enables the EV Counter Interrupt. TCI will not be set. Note that the counter must be enabled by a GSEC command.

SPCNI—Stop Counter Interrupts
(Command = 0F0H)

The 8292 will not generate an EV interrupt when the counter reaches 0. Note that the counter will continue counting. TCI will not be set.

SREM—Set Interface to Remote Control
(Command = 0F8H)

If the 8292 is the System Controller, it will set REN and TCI TRUE. Otherwise it only sets the User Error Flag.

SLOC—Set Interface to Local Mode
(Command = 0F7H)

If the 8292 is the System Controller, it will set REN FALSE and TCI TRUE. Otherwise, it only sets the User Error Flag.

EXPP—Execute Parallel Poll
(Command = 0F5H)

If not Controller-in-Charge, the 8292 will treat this as a NOP and does not set TCI. If it is the Controller-in-Charge then it sets IDY (EOI & ATN) TRUE and generates a local DAV pulse (that never reaches the GPIB because of gates in the 8293). If the 8291 is configured as a listener, it will capture the Parallel Poll Response byte in its data register. TCI is not generated, the CPU must detect the BI (Byte In) from the 8291. The 8292 will be ready to accept another command before the BI occurs; therefore the 8291's BI serves as a task complete indication.

GTSB—Go To Standby (Command = 0F6H)

If the 8292 is not the Controller-in-Charge, it will treat this command as a NOP and does not set TCI TRUE. Otherwise, it goes to Controller Standby State (CSBS), sets ATN FALSE and TCI TRUE. This command is used as part of the Send, Receive, Transfer and Serial Poll System commands (see next section) to allow the addressed talker to send data/status.

If the data transfer does not start within the specified Time-Out, the 8292 sets TOUT2 TRUE in the Error Flag Register and sets SPI (if enabled). The controller continues waiting for a new command. The CPU must decide to wait longer or to regain control and take corrective action.

GSEC—Go To Standby and Enable Counting
(Command = 0F4H)

This command does the same things as GTSB but also initializes the event counter to the value previously stored in the Event Counter Register (default value is 256) and enables the counter. One may wire the count input to $\overline{\text{NDAC}}$ to count bytes. When the counter reaches zero, it sets EV (and SPI if enabled) in Interrupt Status and will set EV every 256 bytes thereafter. Note that there is a potential loss of count information if the CPU does not respond to the EV/SPI before another 256 bytes have been transferred. TCI will be set at the end of the command.

TCSY—Take Control Synchronously
(Command = 0FDH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it waits for the proper handshake state and sets ATN TRUE. The 8292 will set TOUT3 if the handshake never assumes the correct state and will remain in this command until the handshake is proper or a RSTI command is issued. If the 8292 successfully takes control, it sets TCI TRUE.

This is the normal way to regain control at the end of a Send, Receive, Transfer or Serial Poll System Command. If TCSY is not successful, then the controller must try TCAS (see warning below).

TCAS—Take Control Asynchronously
(Command = 0FCH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it arbitrarily sets ATN TRUE and ECI TRUE. Note that this action may cause devices on the bus to lose a data byte or cause them to interpret a data byte as a command byte. Both Actions can result in anomalous behavior. TCAS should be used only in emergencies. If TCAS fails, then the System Controller will have to issue an ABORT to clean things up.

GIDL—Go to Idle (Command = 0F1H)

If the 8292 is not the Controller in Charge and Active, then it treats this command as a NOP and does not set TCI. Otherwise, it sets ATN FALSE, becomes Not Controller in Charge, and sets TCI TRUE. This command is used as part of the Pass Control System Command.

TCNTR—Take (Receive) Control
(Command = 0FAH)

If the 8292 is not Idle, then it treats this command as a NOP and does not set TCI. Otherwise, it waits for the current Controller-in-Charge to set ATN FALSE. If this does not occur within the specified Time Out, the 8292 sets TOUT1 in the Error Flag Register and sets SPI (if enabled). It will not proceed until ATN goes false or it receives an RSTI command. Note that the Controller in Charge must previously have sent this controller (via the 8291's command pass through register) a Pass Control message. When ATN goes FALSE, the 8292 sets CIC, ATN and TCI TRUE and becomes Active.

SOFTWARE DRIVER OUTLINE

The set of system commands discussed below is shown in Figure 14. These commands are implemented in software routines executed by the main CPU.

The following section assumes that the Controller is the System Controller and will not Pass Control. This is a valid assumption for 99+% of all controllers. It also assumes that no DMA or Interrupts will be used. SYC (System Control Input) should not be changed after Power-on in any system—it adds unnecessary complexity to the CPU's software.

In order to use polling with the 8292 one must enable TCI but not connect the pin to the CPU's interrupt pin. TCI must be readable by some means. In this application example it is connected to bit 1 port 6FH on the ZT7488/18. In addition, the other three 8292 interrupt lines and the 8291 interrupt are also on that port (SPI-Bit 2, $\overline{\text{IBFI}}$-Bit 4, OBFI-Bit 3, 8291 INT-Bit 0).

These drivers assume that only primary addresses will be used on the GPIB. To use secondary addresses, one must modify the test for valid talk/listen addresses (range macro) to include secondaries.

INIT	INITIALIZATION
Talker/Listener	
SEND	SEND DATA
RECV	RECEIVE DATA
XFER	TRANSFER DATA
Controller	
TRIG	GROUP EXECUTE TRIGGER
DCLR	DEVICE CLEAR
SPOL	SERIAL POLL
PPEN	PARALLEL POLL ENABLE
PPDS	PARALLEL POLL DISABLE
PPUN	PARALLEL POLL UNCONFIGURE
PPOL	PARALLEL POLL
PCTL	PASS CONTROL
RCTL	RECEIVE CONTROL
SRQD	SERVICE REQUESTED
System Controller	
REME	REMOTE ENABLE
LOCL	LOCAL
IFCL	ABORT/INTERFACE CLEAR

Figure 14. Software Drive Routines

Initialization

8292—Comes up in Controller Active State when SYC is TRUE. The only initialization needed is to enable the TCI interrupt mask. This is done by writing 0A0H to Port 68H.

8291—Disable both the major and minor addresses because the 8291 will never see the 8292's commands/addresses (refer to earlier hardware discussion). This is done by writing 60H and 0E0H to Port 66H.

Set Address Mode to Talk-only by writing 80H to Port 64H.

Set internal counter to 3 MHz to match the clock input coming from the 8085 by writing 23H to Port 65H. High speed mode for the handshakes will not be used here even though the hardware uses three-state drivers.

No interrupts will be enabled now. Each routine will enable the ones it needs for ease of polling operation. The INT bit may be read through Port 6FH. Clear both interrupt mask registers.

Release the chip's initialization state by writing 0 to Port 65H.

```
INIT:
 Enable-8292                            ;Set up In. pins for Port 6FH
  Enable TCI                            ;Task complete must be on
 Enable-8291
  Disable major address                 ;In controller usage, the 8291
  Disable minor address                 ;Is set to talk only and/or listen only
  ton                                   ;Talk only is our rest state
  Clock frequency                       ;3 MHz in this ap note example
  All interrupts off
  Immediate execute pon                 ;Releases 8291 from init. state
```

Talker/Listener Routines

SEND DATA

SEND <listener list pointer> <count> <EOS> <data buffer pointer>

This system command sends data from the CPU to one or more devices. The data is usually a string of ASCII characters, but may be binary or other forms as well. The data is device-specific.

My Talk Address (MTA) must be output to satisfy the GPIB requirement of only one talker at a time (any other talker will stop when MTA goes out). The MTA is not needed as far as the 8291 is concerned—it will be put into talk-only mode (ton).

This routine assumes a non-null listener list in that it always sends Univeral Unlisten. If it is desired to send data to the listeners previously addressed, one could add a check for a null list and not send UNL. Count must be 255 or less due to an 8 bit register. This routine also always uses an EOS character to terminate the string output; this could easily be eliminated and rely on the count. Items in brackets ( ) are optional and will not be included in the actual code in Appendix A.

```
SEND:
  Output-to-8291 MTA, UNL            ;We will talk, nobody listen
  Put EOS into 8291                  ;End of string compare character
  While 20H ≤ listener ≤ 3EH         ;GPIB listen addresses are
   output-to-8291 listener           ;"space" thru ">" ASCII
   Increment listen list pointer     ;Address all listeners
  Output-to-8292 GTSB                ;8292 stops asserting ATN, go to standby
  Enable-8291
   Output EOI on EOS sent            ;Send EOI along with EOS character.
  If count < > 0 then
   While not (end or count = 0)      ;Wait for EOS or end of count
    (could check tout 2 here)        ;Optionally check for stuck bus-tout 2
     Output-to-8291 data             ;Output all data, one byte at a time
     Increment data buffer pointer   ;8085 CREG will count for us
     Decrement count
  Output-to-8292 TCSY                ;8292 asserts ATN, take control sync.
  (If tout3 then take control async) ;If unable to take control sync.
  Enable 8291                        ;Restore 8291 to standard condition
   No output EOI on EOS sent
  Return
```

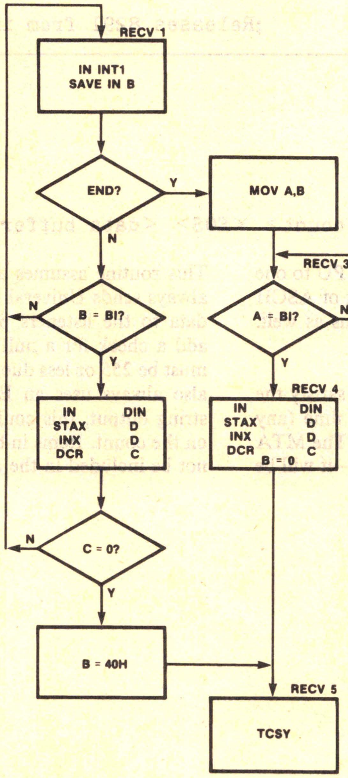


Figure 15. Flowchart for Receive Ending Conditions

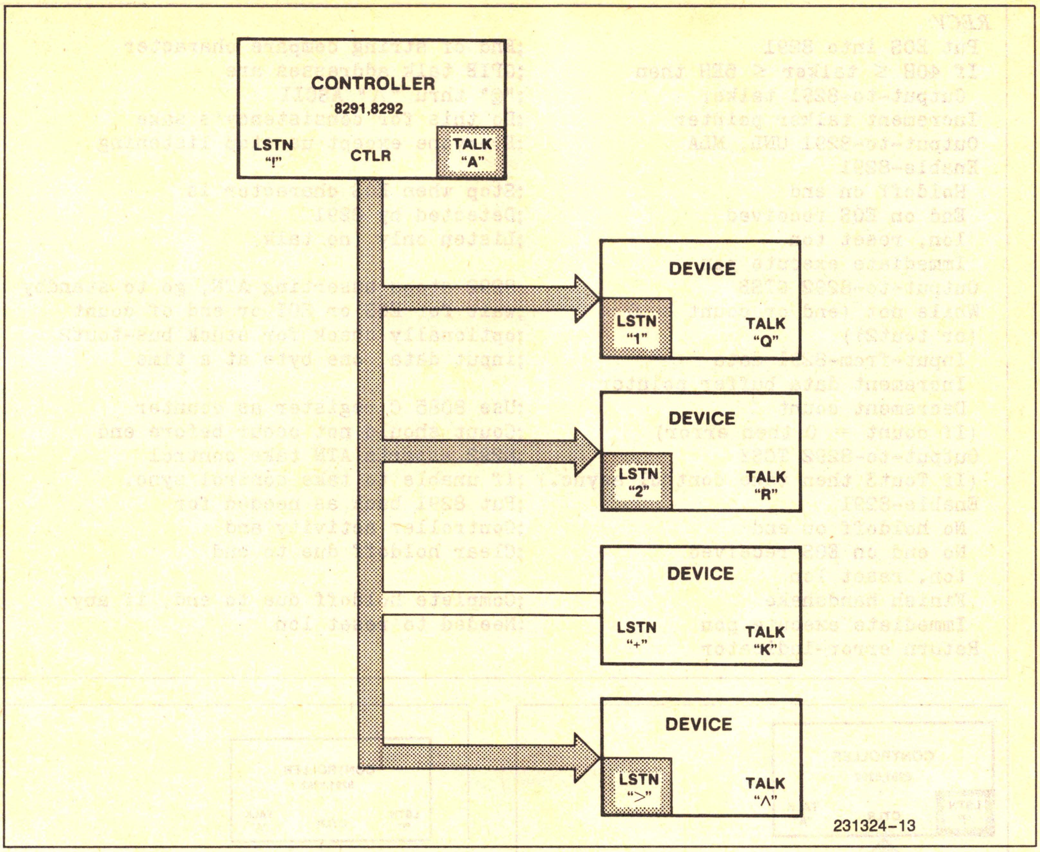


Figure 16. SEND to "1", "2", ">"; "ABCD"; EOS = "D"

RECEIVE DATA

RECV <talker> <count> <EOS> <data buffer pointer>

This system command is used to input data from a device. The data is typically a string of ASCII characters.

This routine is the dual of SEND. It assumes a new talker will be specified, a count of less than 257, and an EOS character to terminate the input. EOI received will also terminate the input. Figure 15 shows the flow chart for the RECV ending conditions. My Listen Address (MLA) is sent to keep the GPIB transactions totally regular to facilitate analysis by a GPIB logic analyzer like the Ziatech ZT488. Otherwise, the bus would appear to have no listener even though the 8291 will be listening.

Note that although the count may go to zero before the transmission ends, the talker will probably be left in a strange state and may have to be cleared by the controller. The count ending of RECV is therefore used as an error condition in most situations.

```
RECV:
 Put EOS into 8291                          ;End of string compare character
 If 40H ≤ talker ≤ 5EH then                 ;GPIB talk addresses are
  Output-to-8291 talker                     ;"@" thru "∧" ASCII
 Increment talker pointer                   ;Do this for consistency's sake
 Output-to-8291 UNL, MLA                    ;Everyone except us stop listening
 Enable-8291
  Holdoff on end                            ;Stop when EOS character is
  End on EOS received                       ;Detected by 8291
  lon, reset ton                            ;Listen only (no talk)
  Immediate execute pon
 Output-to-8292 GTSB                        ;8292 stops asserting ATN, go to standby
 While not (end or count = 0                ;wait for EOS or EOI or end of count
 (or tout2))                                ;optionally check for stuck bus-tout2
  Input-from-8291 data                      ;input data, one byte at a time
  Increment data buffer pointer
  Decrement count                           ;Use 8085 C register as counter
 (If count = 0 then error)                  ;Count should not occur before end
 Output-to-8292 TCSY                        ;8292 asserts ATN take control
 (If Tout3 then take control async.)        ;If unable to take control sync.
 Enable-8291                                ;Put 8291 back as needed for
  No holdoff on end                         ;Controller activity and
  No end on EOS received                    ;Clear holdoff due to end
  ton, reset lon
  Finish handshake                          ;Complete holdoff due to end, if any
  Immediate execute pon                     ;Needed to reset lon
 Return error-indicator
```

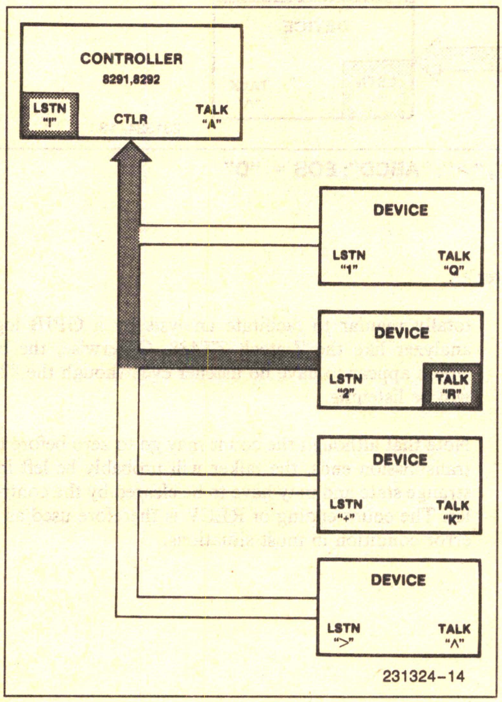


Figure 17. RECV from "R"; EOS = 0DH

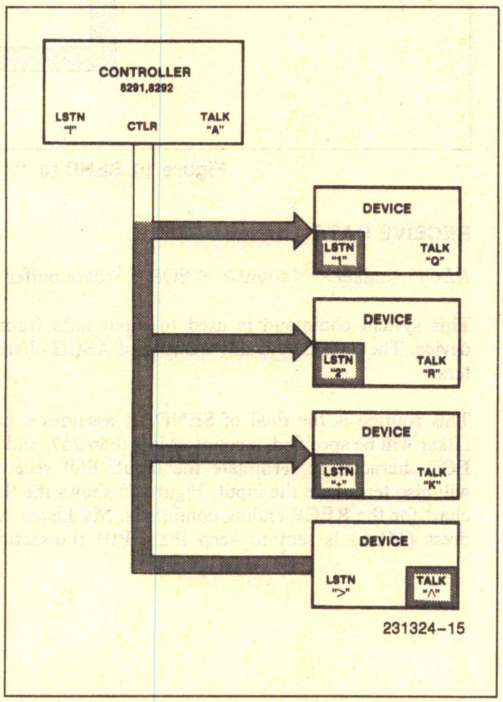


Figure 18. XFER from "∧" to "1", "2", "+"; EOS = 0DH

TRANSFER DATA

XFER <Talker> <Listener list> <EOS>

This system command is used to transfer data from a talker to one or more listeners where the controller does not participate in the transfer of the ASCII data. This is accomplished through the use of the 8291's continuous acceptor handshake mode while in listen-only.

This routine assumes a device list that has the ASCII talker address as the first byte and the string (one or more) or ASCII listener addresses following. The EOS character or an EOI will cause the controller to take take control synchronously and thereby terminate the transfer.

```
XFER:
 Output-to-8291: Talker, UNL          ;Send talk address and unlisten
 While 20H ≤ listen ≤ 3EH
  Output-to-8291: Listener            ;Send listen address
  Increment listen list pointer
 Enable-8291
  lon, no ton                         ;Controller is pseudo listener
  Continuous AH mode                  ;Handshake but don't capture data
  End on EOS received                 ;Capture EOS as well as EOI
  Immediate execute PON               ;Initialize the 8291
 Put EOS into 8291                    ;Set up EOS character
 Output-to-8292: GTSB                 ;Go to standby
                                      ;8292 waits for EOS or EOI and then
 Upon end (or tout2) then
  Take control synchronously          ;Regains control
 Enable-8291                          ;Go to Ready for Data
  Finish handshake
  Not continuous AH mode
  Not END on EOS received
  ton
  Immediate execute pon
 Return
```

Controller

GROUP EXECUTE TRIGGER

TRIG <Listener list>

This system command causes a group execute trigger (GET) to be sent to all devices on the listener list. The intended use is to synchronize a number of instruments.

```
TRIG:
 Output-to-8291 UNL                   ;Everybody stop listening
 While 20H ≤ listener ≤ 3EH           ;Check for valid listen address
  Output-to-8291 Listener             ;Address each listener
  Increment listen list pointer       ;Terminate on any non-valid character
 Output-to-8291 GET                   ;Issue group execute trigger
 Return
```

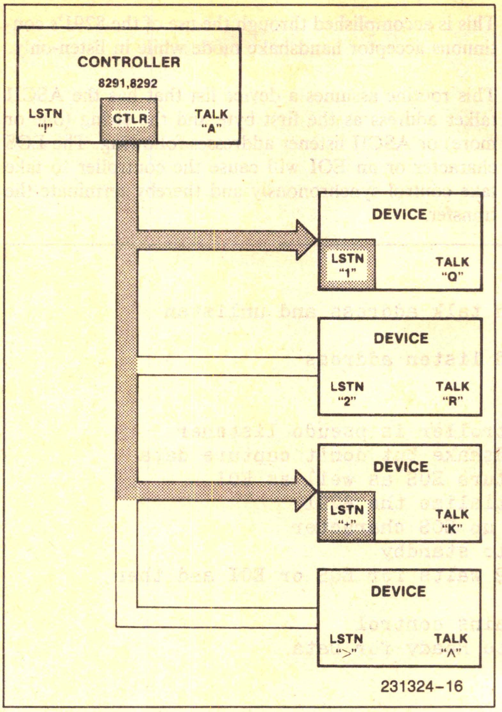


Figure 19. TRIG "1", "+"

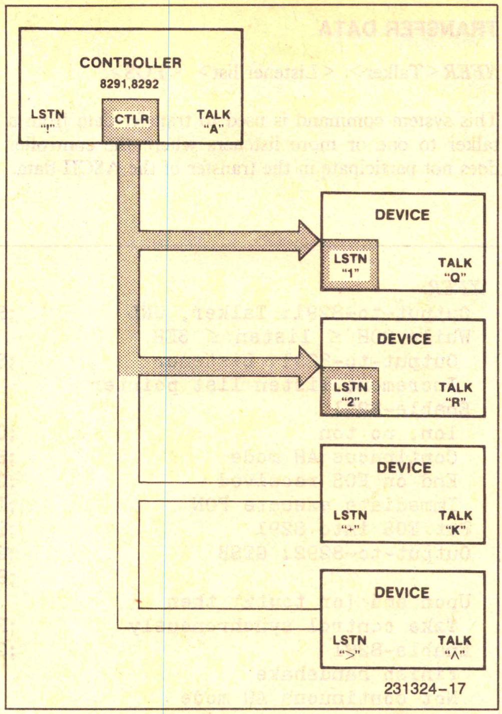


Figure 20. DCLR "1", "2"

DEVICE CLEAR

DCLR <Listener list>

This system command causes a device clear (SDC) to be sent to all devices on the listener list. Note that this is not intended to clear the GPIB interface of the device, but should clear the device-specific logic.

```
DCLR:
 Output-to-8291 UNL               ;Everybody stop listening
 While 20H ≤ listener ≤ 3EH       ;Check for valid listen address
  Output-to-8291 Listener         ;Address each listener
  Increment listen list pointer   ;Terminate on any non-valid character
 Output-to-8291 SDC               ;Selective device clear
 Return
```

SERIAL POLL

SPOL <Talker list> <status buffer pointer>

This system command sequentially addresses the designated devices and receives one byte of status from each. The bytes are stored in the buffer in the same order as the devices appear on the talker list. MLA is output for completeness.

```
SPOL:
 Output-to-8291 UNL, MLA, SPE         ;Unlisten, we listen, serial poll enable
                                      ;Only one byte of serial poll
                                      ;Status wanted from each talker

 While 40H ≤ talker ≤ 5 EH            ;Check for valid transfer
  Output-to-8291 talker               ;Address each device to talk
  Increment talker list pointer       ;One at a time
  Enable-8291
   lon, reset ton                     ;Listen only to get status
   Immediate execute pon              ;This resets ton
  Output-to-8292 GTSB                 ;Go to standby
  Wait for data in (BI)               ;Serial poll status byte into 8291
  Output-to-8292 TCSY                 ;Take control synchronously
  Input-from-8291 data                ;Actually get data from 8291
  Increment buffer pointer
  Enable 8291
   ton, reset lon
   Immediate execute pon              ;Reset lon
  Output-to-8291 SPD                  ;Send serial poll disable after all
                                       devices polled
  Return
```

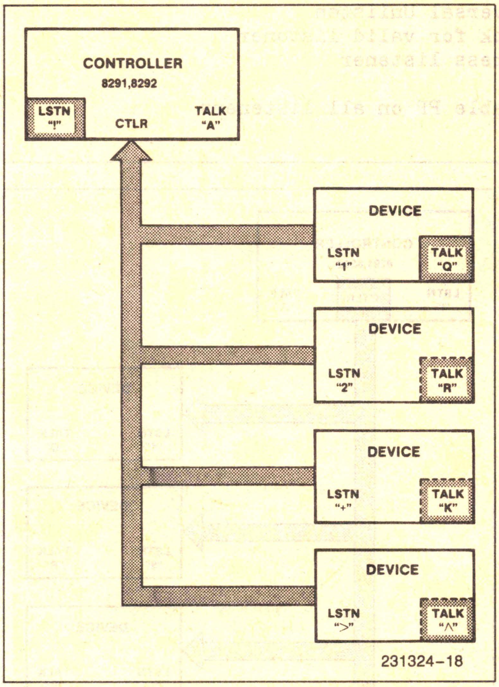


Figure 21. SPOL "Q", "R", "K", "Λ"

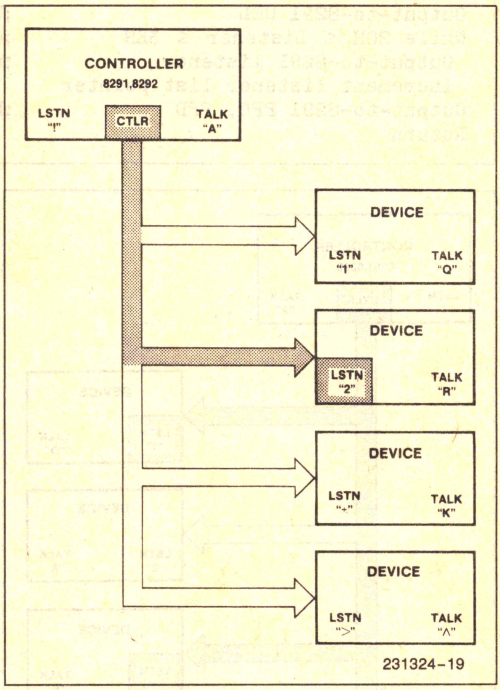


Figure 22. PPEN "2"; $iP_3P_2P_1$ = 0111B

PARALLEL POLL ENABLE

PPEN <Listener list> <Configuration Buffer pointer>

This system command configures one or more devices to respond to Parallel Poll, assuming they implement subset PP1. The configuration information is stored in a buffer with one byte per device in the same order as

devices appear on the listener list. The configuration byte has the format XXXXIP3P2P1 as defined by the IEEE Std. P3P2P1 indicates the bit # to be used for a response and I indicates the assertion value. See Sec. 2.9.3.3 of the Std. for more details.

```
PPEN:
 Output-to-8291 UNL                    ;Universal unlisten
 While 20H ≤ Listener ≤ 3EH            ;Check for valid listener
  Output-to-8291 listener              ;Stop old listener, address new
  Output-to-8291 PPC, (PPE or data)    ;Send parallel poll info
  Increment listener list pointer      ;Point to next listener
  Increment buffer pointer             ;One configuration byte per listener
 Return
```

PARALLEL POLL DISABLE

PPDS<listener list>

This system command disables one or more devices from responding to a Parallel Poll by issuing a Parallel Poll Disable (PPD). It does not deconfigure the devices.

```
PPDS:
 Output-to-8291 UNL                    ;Universal Unlisten
 While 20H ≤ Listener ≤ 3EH            ;Check for valid listener
  Output-to-8291 listener              ;Address listener
  Increment listener list pointer
 Output-to-8291 PPC, PPD               ;Disable PP on all listeners
 Return
```

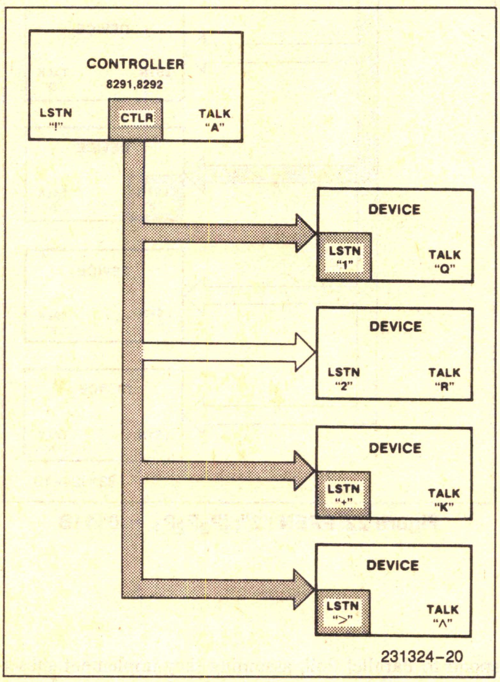


Figure 23. PPDS "1", "+", ">"

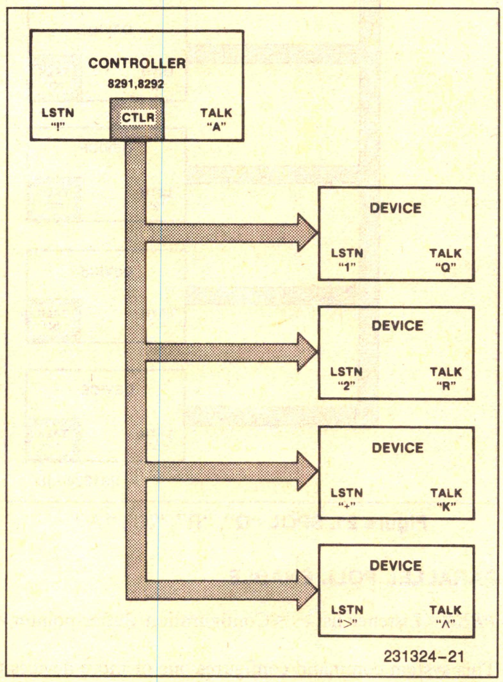


Figure 24. PPUN

PARALLEL POLL UNCONFIGURE

PPUN

This system command deconfigures the Parallel Poll response of all devices by issuing a Parallel Poll Unconfigure message.

```
PPUN:
  Output-to-8291 PPU                        ;Unconfigure all parallel poll
  Return
```

CONDUCT A PARALLEL POLL

PPOL

This system command causes the controller to conduct a Parallel Poll on the GPIB for approximately 12.5 μsec (at 6 MHz). Note that a parallel poll does not use the handshake; therefore, to ensure that the device knows whether or not its positive response was observed by the controller, the CPU should explicitly acknowledge each device by a device-dependent data string. Otherwise, the response bit will still be set when the next Parallel Poll occurs. This command returns one byte of status.

```
PPOL:
  Enable-8291
   lon                                      ;Listen only
   Immediate execute pon                    ;This resets ton
  Output-to-8292 EXPP                       ;Execute parallel poll
  Upon BI                                   ;When byte is input
   Input-from-8291 data                     ;Read it
  Enable-8291
   ton                                      ;Talk only
   Immediate execute pon                    ;This resets lon
  Return Data (status byte)
```

PASS CONTROL

PCTL <talker>

This system command allows the controller to relinquish active control of the GPIB to another controller. Normally some software protocol should already have informed the controller to expect this, and under what conditions to return control. The 8291 must be set up to become a normal device and the CPU must handle all commands passed through, otherwise control cannot be returned (see Receive Control below). The controller will go idle.

```
PCTL:
  If 40H ≤ talker ≤ 5EH then
   if talker < > MTA then                   ;Cannot pass control to myself
    output-to-8291 talker, TCT              ;Take control message to talker
    Enable-8291                             ;Set up 8291 as normal device
     not ton, not lon
     Immediate execute pon                  ;Reset ton and lon
     My device address, mode 1              ;Put device number in Register 6
     Undefined command pass through         ;Required to receive control
     (Parallel Poll Configuration)          ;Optional use of PP
   Output-to-8292 GIDL                      ;Put controller in idle
  Return
```

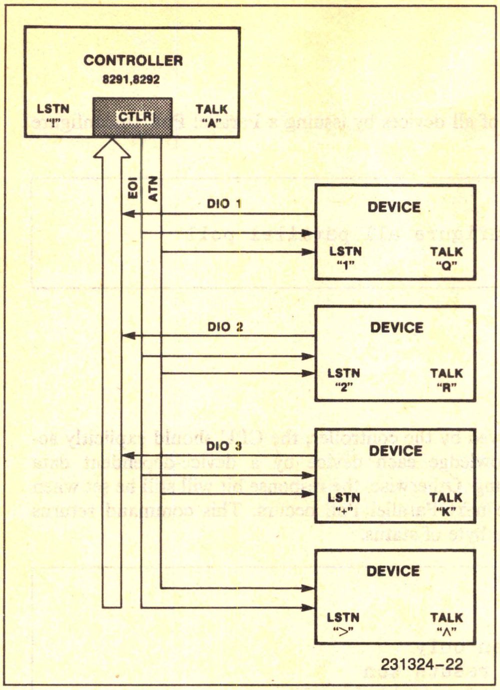


Figure 25. PPOL

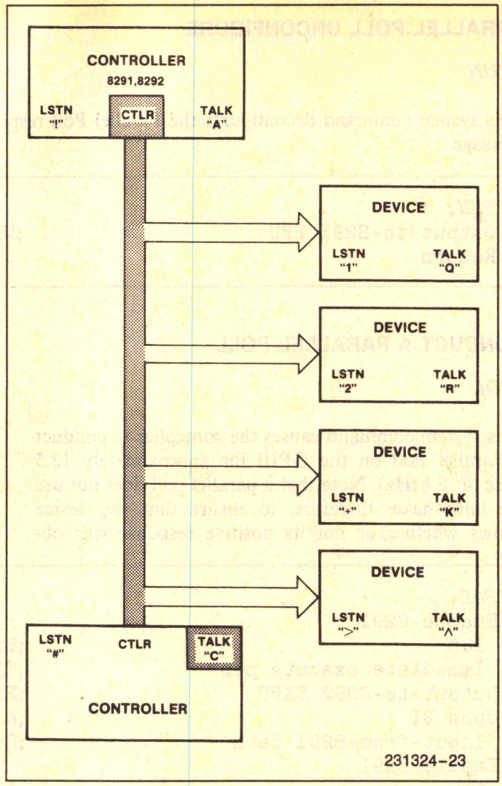


Figure 26. PCTL "C"

RECEIVE CONTROL

RCTL

This system command is used to get control back from the current controller-in-charge if it has passed control to this inactive controller. Most GPIB systems do not use more than one controller and therefore would not need this routine.

To make passing and receiving control a manageable event, the system designer should specify a protocol whereby the controller-in-charge sends a data message to the soon-to-be-active controller. This message should give the current state of the system, why control is being passed, what to do, and when to pass control back. Most of these issues are beyond the scope of this Ap Note.

```
RCTL:
 Upon CPT                             ;Wait for command pass through bit in 8291
  If (command=TCT) then               ;If command is take control and
   If TA then                         ;We are talker addressed
    Enable-8291
     Disable major device number      ;Controller will use ton and lon
     ton                              ;Talk only mode
     Mask off interrupts
     Immediate execute pon
    Output-to-8292 TCNTR              ;Take (receive) control
    Enable-8291
     Valid command                    ;Release handshake
    Return valid
   Else
    Enable-8291
     Invalid command                  ;Not talker addr. so TCT not for us
  Else
   Enable-8291
    Invalid command                   ;Not TCT, so we don't care
 Return invalid
```

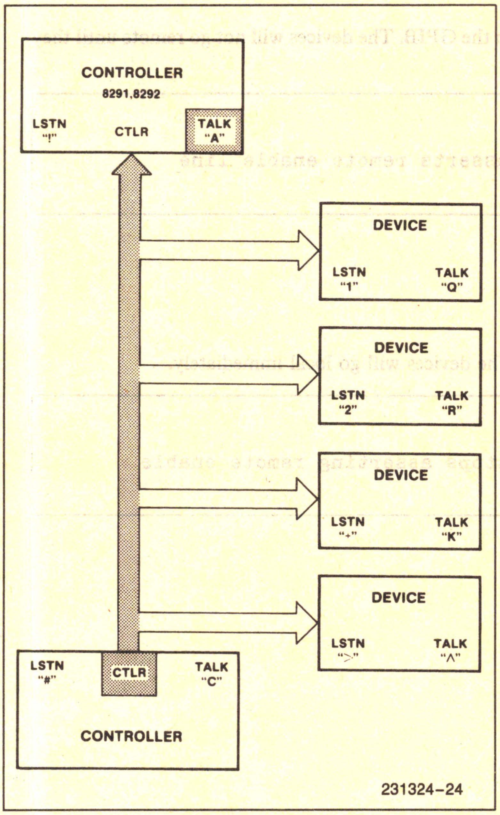


Figure 27. RCTL

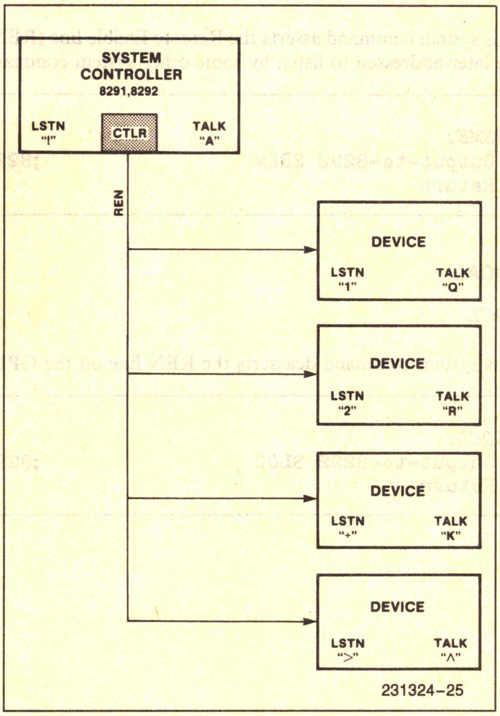


Figure 28. REME

SERVICE REQUEST

SRQD

This system command is used to detect the occurrence of a Service Request on the GPIB. One or more devices may assert SRQ simultaneously, and the CPU would normally conduct a Serial Poll after calling this routine to determine which devices are SRQing.

```
SRQD:
 If SRQ then                         ;Test 92 status bit
  Output-to-8292 IACK.SRQ            ;Acknowledge it
  Return SRQ
  Else return no SRQ
```

System Controller

REMOTE ENABLE

REME

This system command asserts the Remote Enable line (REN) on the GPIB. The devices will not go remote until they are later addressed to listen by some other system command.

```
REME:
 Output-to-8292 SREM                 ;8292 asserts remote enable line
 Return
```

LOCAL

LOCL

This system command deasserts the REN line on the GPIB. The devices will go local immediately.

```
LOCL:
 Output-to-8292 SLOC                 ;8292 stops asserting remote enable
 Return
```

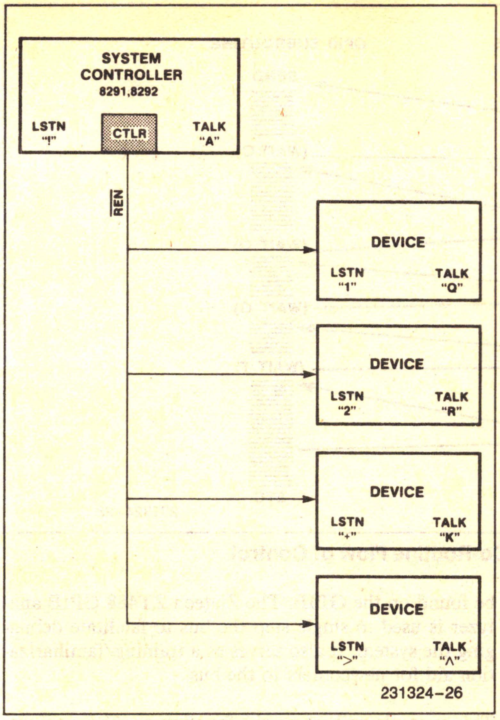


Figure 29. LOCL

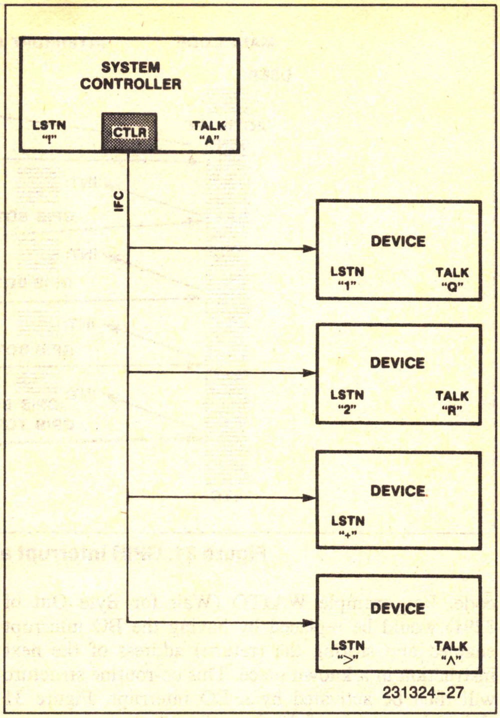


Figure 30. IFCL

INTERFACE CLEAR/ABORT

IFCL

This system command asserts the GPIB's Interface Clear (IFC) line for at least 100 microseconds. This causes all interface logic in all devices to go to a known state. Note that the device itself may or may not be reset, too. Most instruments do totally reset upon IFC. Some devices may require a DCLR as well as an IFCL to be completely reset. The (system) controller becomes Controller-in-Charge.

```
IFCL:
  Output-to-8292 ABORT                    ;8292 asserts Interface Clear
  Return                                  ;For 100 microseconds
```

INTERRUPTS AND DMA CONSIDERATIONS

The previous sections have discussed in detail how to use the 8291, 8292, 8293 chip set as a GPIB controller with the software operating in a polling mode and using programmed transfer of the data. This is the simplest mode of use, but it ties up the microprocessor for the duration of a GPIB transaction. If system design constraints do not allow this, then either Interrupts and/or DMA may be used to free up processor cycles.

The 8291 and 8292 provide sufficient interrupts that one may return to do other work while waiting for such things as 8292 Task Completion, 8291 Next Byte In, 8291 Last Byte Out, 8292 Service Request In, etc. The only difficulty lies in integrating these various interrupt sources and their matching routines into the overall system's interrupt structure. This is highly situation-specific and is beyond the scope of this Ap Note.

The strategy to follow is to replace each of the WAIT routines (see Appendix A) with a return to the main code and provide for the corresponding interrupt to bring the control back to the next section of GPIB

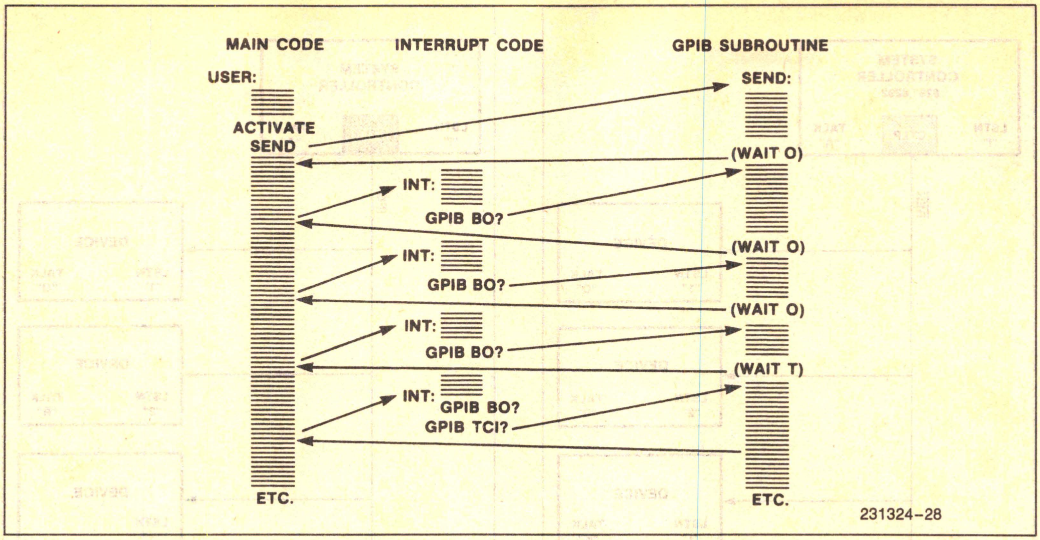


Figure 31. GPIB Interrupt and Co-Routine Flow of Control

code. For example WAITO (Wait for Byte Out of 8291) would be replaced by having the BO interrupt enabled and storing the (return) address of the next instruction in a known place. This co-routine structure will then be activated by a BO interrupt. Figure 31 shows an example of the flow of control.

DMA is also useful in relieving the processor if the average length of a data buffer is long enough to overcome the extra time used to set up a DMA chip. This decision will also be a function of a data rate of the instrument. The best strategy is to use the DMA to handle only the data buffer transfers on SEND and RECV and to do all the addressing and control just as shown in the driver descriptions.

Another major reason for using a DMA chip is to increase the data rate and therefore increase the overall transaction rate. In this case the limiting factor becomes the time used to do the addressing and control of the GPIB using software like that in Appendix A. The data transmission time becomes insignificant at DMA speeds unless extremely long buffers are used.

Refer to Figure 11 for a typical DMA and interrupt based design using the 8291, 8292, 8293. A system like this can achieve a 250K byte transfer rate while under DMA control.

APPLICATION EXAMPLE

This section will present the code required to operate a typical GPIB instrument set up as shown in Figure 32. The HP5328A universal counter and the HP3325 function generator are typical of many GPIB devices; however, there are a wide variety of software protocols to be found on the GPIB. The Ziatech ZT488 GPIB analyzer is used to single step the bus to facilitate debugging the system. It also serves as a training/familiarization aid for newcomers to the bus.

This example will set up the function generator to output a specific waveform, frequency and amplitude. It will then tell the counter to measure the frequency and Request Service (SRQ) when complete. The program will then read in the data. The assembled source code will be found at the end of Appendix A.

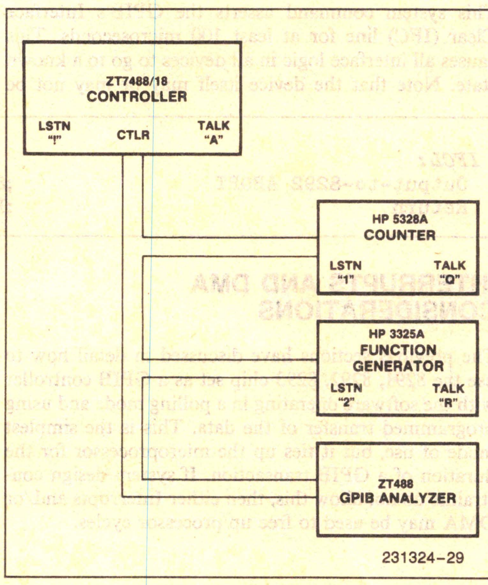


Figure 32. GPIB Example Configuration

```
SEND
 LSTN: "2", COUNT: 15, EOS: ODH, DATA: "FU1FR37KHAM2VO (CR)"
  ;SETS UP FUNCTION GEN. TO 37 KHZ SINE, 2 VOLTS PP
  ;COUNT EQUAL TO # CHAR IN BUFFER
  ;EOS CHARACTER IS (CR) = ODH = CARRIAGE

SEND
 LSTN: "1", COUNT: 6, EOS: "T" DATA: "PR4G7T"
  ;SETS UP COUNTER FOR P:INITIALIZE, F4: FREQ CHAN A
  ;         G7:0.1 HZ RESOLUTION, T:TRIGGER AND SRQ
  ;COUNT IS EQUAL TO # CHAR

WAIT FOR SRQ
SPOL TALK: "Q", DATA: STATUS 1
  ;CLEARS THE SRO_IN THIS EXAMPLE ONLY FREQ CTR ASSERTS SRQ

RECV TALK: "Q", COUNT: 17, EOS: OAH,
 DATA: "+   37000.0E+0" (CR) (LF)
  ;GETS 17 BYTES OF DATA FROM COUNTER
  ;COUNT IS EXACT BUFFER LENGTH
  ;DATA SHOWN IS TPYICAL HP5328A READING THAT WOULD BE RECEIVED
```

CONCLUSION

This Application Note has shown a structured way to view the IEEE 488 bus and has given typical code sequences to make the Intel 8291, 8292, and 8293's behave as a controller of the GPIB. There are other ways to use the chip set, but whatever solution is chosen, it must be integrated into the overall system software.

The ultimate reference for GPIB questions is the IEEE Std 488 -1978 which is available from IEEE, 345 East 47th St., New York, NY, 10017. The ultimate reference for the 8292 is the source listing for it (remember it's a pre-programmed UPI-41A) which is available from INSITE, Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

APPENDIX A

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0
GPIB CONTROLLER SUBROUTINES

LOC  OBJ          LINE        SOURCE STATEMENT

                      $TITLE('GPIB CONTROLLER SUBROUTINES')
                      ;
                      ; GPIB CONTROLLER SUBROUTINES

                      ;
                      ;        for Intel 8291, 8292 on ZT 7488/18
                      ;        Bert Forbes, Ziatech Corporation
                      ;        2410 Broad Street
                      ;        San Luis Obispo, CA, USA 93401
                      ;
                      ;
                      ;        General Definitions & Equates
                      ;        8291 Control Values
                      ;
1000                           ORG     1000H    ; For ZT7488/18 w/8085
                      ;
0060                  PRT91    EQU     60H      ;8291 Base Port #
                      ;
                      ;        Reg #0 Data in & Data out
0060                  DIN      EQU     PRT91+0  ;91 Data in reg
0060                  DOUT     EQU     PRT91+0  ;91 Data out reg
                      ;
                      ;        Reg # 1 Interrupt 1 Constants
0061                  INT1     EQU     PRT91+1  ;INT Reg 1
0061                  INTM1    EQU     PRT91+1  ;INT Mask Reg. 1
0002                  BOM      EQU     02       ;91 BO INTRP Mask
0001                  BIM      EQU     01       ;91 BI INTRP Mask
0010                  ENDMK    EQU     10H      ;91 END INTRP Mask
0080                  CPT      EQU     80H      ;91 command pass thru int bit
                      ;
                      ;        Reg #2 Interrupt 2
0062                  INT2     EQU     PRT91+2
                      ;
                      ;        Reg #4 Address Mode Constants
0064                  ADRMD    EQU     PRT91+4  ;91 address mode register #
0080                  TON      EQU     80H      ;91 talk only mode & not listen only
0040                  LON      EQU     40H      ;91 listen only & not ton
00C0                  TLON     EQU     0C0H     ;91 talk & listen only
0001                  MODE1    EQU     01       ;mode 1 addressing for device

                      ;        Reg #4            (Read)  Address Status Register
0064                  ADRST    EQU     PRT91+4  ;reg #4
0020                  EOIST    EQU     20H
0002                  TA       EQU     2
0001                  LA       EQU     1        ;listener active
                      ;
                      ;        Reg #5  (Write) Auxillary Mode Register
0065                  AUXMD    EQU     PRT91+5  ;91 auxillary mode register #
0023                  CLKRT    EQU     23H      ;91 3 Mhz clock input
```

231324-30

```
0003   FNHSK   EQU     03       ;91 fininsh handshake command
0006   SDEOI   EQU     06       ;91 send EOI with next byte
0080   AXRA    EQU     80H      ;91 aux. req A pattern
0001   HOHSK   EQU      1       ;91 hold off handshake on all bytes
0002   HOEND   EQU      2       ;91 hold off handshake on end
0003   CAHCY   EQU      3       ;91 continuous AH cycling
0004   EDEOS   EQU      4       ;91 end on EOS received
0008   EOIS    EQU      8       ;91 output EOI on EOS sent
000F   VSCMD   EQU     0FH      ;91 valid command pass through
0007   NVCMD   EQU     07H      ;91 invalid command pass through
00A0   AXRB    EQU     0A0H     ;Aux. reg. B pattern
0001   CPTEN   EQU     01H      ;command pass thru enable
       ;
       ;       Reg #5  (Read)
0065   CPTRG   EQU     PRT91+5
       ;
       ;       Reg #6  Address 0/1 reg. constants
0066   ADR01   EQU     PRT91+6
0060   DTDL1   EQU     60H      ;Disable major talker & listener
00E0   DTDL2   EQU     0E0H     ;Disable minor talker & listener
       ;
       ;       Reg #7  EOS      Character Register
0067   EOSR    EQU     PRT91+7
       ;
       ;
       ;       8292    CONTROL VALUES
       ;
       ;
       ;
0068   PRT92   EQU     PRT91+8  ;8292 Base Port # (CS7)
       ;
0068   INTMR   EQU     PRT92+0  ;92 INTRP Mask Reg
00A0   INTM    EQU     0A0H     ;TCI
       ;
0068   ERRM    EQU     PRT92+0  ;92 Error Mask Reg
0001   TOUT1   EQU     01       ;92 Time Out for Pass Control
0002   TOUT2   EQU     02       ;92 Time Out for Standby
0004   TOUT3   EQU     04       ;92 Time Out for Take Control Sync
0068   EVREG   EQU     PRT92+0  ;92 Event Counter Pseudo Reg
0068   TOREG   EQU     PRT92+0  ;92 Time Out Pseudo Reg
       ;
0069   CMD92   EQU     PRT92+1  ;92 Command Register
       ;
0069   INTST   EQU     PRT92+1  ;92 Interrupt Status Reg
0010   EVBIT   EQU     10H      ;Event Counter Bit
0002   IBFBT   EQU     02       ;Input Buffer Full Bit
0020   SRQBT   EQU     20H      ;Seq bit
       ;
0068   ERFLG   EQU     PRT92+0  ;92 Error Flag Pseudo Reg
0068   CLRST   EQU     PRT92+0  ;92 Controller Status Pseudo Reg
0068   BUSST   EQU     PRT92+0  ;92 GPIB (Bus) Status Pseudo Reg
0068   EVCST   EQU     PRT92+0  ;92 Event Counter Status Pseudo Reg
0068   TOST    EQU     PRT92+0  ;92 Time Out Status Pseudo Reg
       ;
       ;       8292    OPERATION COMMANDS
       ;
       ;
00F0   SPCNI   EQU     0F0H     ;Stop Counter Interrupts
00F1   GIDL    EQU     0F1H     ;Go to idle
00F2   RSET    EQU     0F2H     ;Reset
00F3   RSTI    EQU     0F3H     ;Reset Interrupts
00F4   GSEC    EQU     0F4H     ;Goto standby, enable counting
00F5   EXPP    EQU     0F5H     ;Execute parallel poll
00F6   GTSB    EQU     0F6H     ;Goto standby
00F7   SLOC    EQU     0F7H     ;Set local mode
00F8   SREM    EQU     0F8H     ;Set interface to remote
00F9   ABORT   EQU     0F9H     ;Abort all operation, clear interface
00FA   TCNTR   EQU     0FAH     ;Take control (Receive control)
00FC   TCASY   EQU     0FCH     ;Take control asyncronously
00FD   TCSY    EQU     0FDH     ;Take control syncronously
00FE   STCNI   EQU     0FEH     ;Start counter interrupts
       ;
       ;
```

231324-31

```
                ;       8292    UTILITY COMMANDS
                ;
                ;
ØØE1            WOUT    EQU     ØE1H        ;Write to timeout reg
ØØE2            WEVC    EQU     ØE2H        ;Write to event counter
ØØE3            REVC    EQU     ØE3H        ;Read event counter status
ØØE4            RERF    EQU     ØE4H        ;Read error flag reg
ØØE5            RINM    EQU     ØE5H        ;Read interrupt mask reg
ØØE6            RCST    EQU     ØE6H        ;Read controller status reg
ØØE7            RBST    EQU     ØE7H        ;Read GPIB Bus status reg
ØØE9            RTOUT   EQU     ØE9H        ;Read timeout status reg
ØØEA            RERM    EQU     ØEAH        ;Read error mask reg
ØØØB            IACK    EQU     ØBH         ;Interrupt Acknowledge
                ;
                ;
                ;       PORT F BIT ASSIGNMENTS
                ;
                ;
                ;
ØØ6F            PRTF    EQU     PRT91+ØFH           ;ZT7488 port 6F for interrupts
ØØØ2            TCIF    EQU     Ø2H         ;Task complete interrupt
ØØØ4            SPIF    EQU     Ø4H         ;Special interrupt
ØØØ8            OBFF    EQU     Ø8H         ;92 Output (to CPU) Buffer full
ØØ1Ø            IBFF    EQU     1ØH         ;92 Input (from CPU) Buffer empty
ØØØ1            BOF     EQU     Ø1H         ;91 Int line (BO in this case)
                ;
                ;       GPIB MESSAGES (COMMANDS)
                ;
ØØØ1            MDA     EQU     1           ;My device address is 1
ØØ41            MTA     EQU     MDA+4ØH     ;My talk address is 1 ("A")
ØØ21            MLA     EQU     MDA+2ØH     ;My listen address is 1 ("!")
ØØ3F            UNL     EQU     3FH         ;Universal unlisten
ØØØ8            GET     EQU     Ø8          ;Group Execute Trigger
ØØØ4            SDC     EQU     Ø4H         ;Device Clear
ØØ18            SPE     EQU     18H         ;Serial poll enable
ØØ19            SPD     EQU     19H         ;Serial poll disable
ØØØ5            PPC     EQU     Ø5          ;Parallel poll configure
ØØ7Ø            PPD     EQU     7ØH         ;Parallel poll disable
ØØ6Ø            PPE     EQU     6ØH         ;Parallel poll disable
ØØ15            PPU     EQU     15H         ;Parallel poll unconfigured
ØØØ9            TCT     EQU     Ø9          ;Take control (pass control)

                ;       MACRO DEFINITIONS
                ;
                ;
                ;
                SETF    MACRO               ;Sets flags on A register
                        ORA     A
                        ENDM
                ;
                WAITO   MACRO               ;Wait for last 91 byte to be done
                        LOCAL   WAITL
                WAITL:  IN      INT1        ;Get Intl status
                        ANI     BOM         ;Check for byte out
                        JZ      WAITL       ;If not, try again
                        ENDM                ;until it is
                ;
                ;
                WAITI   MACRO               ;Wait for 91 byte to be input
                        LOCAL   WAITL
                WAITL:  IN      INT1        ;Get INT1 status
                        MOV     B,A         ;Save status in B
                        ANI     BIM         ;Check for byte in
                        JZ      WAITL       ;If not, just try again
                        ENDM                ;until it is
                ;
                WAITX   MACRO               ;Wait for 92's TCI to go false
                        LOCAL   WAITL
                WAITL:  IN      PRTF
                        ANI     TCIF
                        JNZ     WAITL
                        ENDM
                ;
```

231324–32

```
          WAITT   MACRO
                  LOCAL   WAITL
          WAITL:  IN      PRTF      ;Get task complete int,etc.
                  ANI     TCIF      ;Mask it
                  JZ      WAITL     ;Wait for task to be complete
                  ENDM

          RANGE   MACRO   LOWER,UPPER,LABEL
                                    ;Checks for value in range
                                    ;branches to label if not
                                    ;in range. Falls through if
                                    ;lower <= ( (H)(L) ) <= upper.
                                    ;Get next byte.
                  MOV     A,M
                  CPI     LOWER
                  JM      LABEL
                  CPI     UPPER+1
                  JP      LABEL
                  ENDM
          ;
          CLRA    MACRO
                  XRA     A         ;A XOR A =0
                  ENDM
          ;
          ;       All of the following routines have these common
          ;       assumptions about the state of the 8291 & 8292 upon entry
          ;       to the routine and will exit the routine in an identical state.
          ;
          ;
          ;       8291:   BO is or has been set,
          ;               All interrupts are masked off
          ;               TON mode, not LA
          ;               No holdoffs in effect or enabled
          ;               No holdoffs waiting for finish command
          ;
          ;       8292:   ATN asserted (active controller)
          ;               note: RCTL is an exception--- it expects
          ;               to not be active controller
          ;               Any previous task is complete & 92 is
          ;               ready to receive next command.
          ;       8085:   Pointer registers (DE,HL) end one
          ;               beyond last legal entry
          ;**********************************************************
          ;
          ;
          ;       INITIALIZATION ROUTINE
          ;
          ;INPUTS:        None
          ;OUTPUTS:       None
          ;CALLS:         None
          ;DESTROYS:      A,F
          ;
1000 3EA0 INIT:   MVI     A,INTM    ;Enable TCI
1002 D368         OUT     INTMR     ;Output to 92's intr. mask reg
1004 3E60         MVI     A,DTDL1   ;Disable major talker/listener
1006 D366         OUT     ADR01
1008 3EE0         MVI     A,DTDL2   ;Disable minor talker/listener
100A D366         OUT     ADR01
100C 3E80         MVI     A,TON     ;Talk only mode
100E D364         OUT     ADRMD
1010 3E23         MVI     A,CLKRT   ;3 MHZ for delay timer
1012 D365         OUT     AUXMD
                  CLRA
1014 AF           XRA     A         ;A XOR A =0
1015 D361         OUT     INT1
1017 D362         OUT     INT2      ;Disable all 91 mask bits
1019 D365         OUT     AUXMD     ;Immediate execute PON
101B C9           RET
          ;
          ;**********************************************************
          ;
          ;
          ;       SEND ROUTINE
          ;
```

231324-33

```
               ;
               ;
               ;         INPUTS:         HL listener list pointer
               ;                         DE data buffer pointer
               ;                         C  count-- 0 will cause no data to be sent
               ;                         b  EOS character-- software detected
               ;         OUTPUTS:        none
               ;         CALLS:          none
               ;         DESTROYS:       A, C, DE, HL, F
               ;
               ;
               ;
101C 3E41        SEND:    MVI     A,MTA     ;Send MTA to turn off any
101E D360                 OUT     DOUT      ;previous talker
                          WAITO
1020 DB61      +??0001:   IN      INT1      ;Get Intl status
1022 E602      +          ANI     BOM       ;Check for byte out
1024 CA2010    +          JZ      ??0001    ;If not, try again
1027 3E3F                 MVI     A,UNL     ;Send universal unlisten
1029 D360                 OUT     DOUT      ;to stop previous listeners
102B 78                   MOV     A,B       ;Get EOS character
102C D367                 OUT     EOSR      ;Output it to 8291
                                            ;while listener.....
                 SEND1:   RANGE   20H,3EH,SEND2    ;Check next listen address
               +                            ;Checks for value in range
               +                            ;branches to label if not
               +                            ;in range. Falls through if
               +                            ;lower <= ( (H)(L) ) <= upper.
               +                            ;Get next byte.
102E 7E        +          MOV     A,M
102F FE20      +          CPI     20H
1031 FA4710    +          JM      SEND2
1034 FE3F      +          CPI     3EH+1
1036 F24710    +          JP      SEND2
                          WAITO                       ;Wait for previous listener sent
1039 DB61      +??0002:   IN      INT1      ;Get Intl status
103B E602      +          ANI     BOM       ;Check for byte out
103D CA3910    +          JZ      ??0002    ;If not, try again
1040 7E                   MOV     A,M                 ;Get this listener
1041 D360                 OUT     DOUT                ;Output to GPIB
1043 23                   INX     H                   ;Increment listener list pointer
1044 C32E10               JMP     SEND1               ;Loop till non-valid listener
                                                      ;Enable 91 ending conditions
                 SEND2:   WAITO             ;Wait for lstn addr accepted
1047 DB61      +??0003:   IN      INT1      ;Get Intl status
1049 E602      +          ANI     BOM       ;Check for byte out
104B CA4710    +          JZ      ??0003    ;If not, try again
                                            ;WAITO required for early versions
                                            ;of 8292 to avoid GTSB before DAC
104E 3EF6                 MVI     A,GTSB    ;Goto standby
1050 D369                 OUT     CMD92     ;
1052 3E88                 MVI     A,AXRA+EOIS         ;Send EOI with EOS character
1054 D365                 OUT     AUXMD
                          WAITX             ;Wait for TCI to go false
1056 DB6F      +??0004:   IN      PRTF
1058 E602      +          ANI     TCIF
105A C25610    +          JNZ     ??0004
                          WAITT             ;Wait for TCI on GTSB
105D DB6F      +??0005:   IN      PRTF      ;Get task complete int,etc.
105F E602      +          ANI     TCIF      ;Mask it
1061 CA5D10    +          JZ      ??0005    ;Wait for task to be complete

               ;          delete next 3 instructions to make count of 0=256
               ;
1064 79                   MOV     A,C       ;Get count
                          SETF              ;Set flags
1065 B7        +          ORA     A
1066 CA8810               JZ      SEND6     ;If count=0, send no data
1069 1A          SEND3:   LDAX    D         ;Get data byte
106A D360                 OUT     DOUT      ;Output to GPIB
106C B8                   CMP     B         ;Test EOS ...this is faster
                                            ;and uses less code than using
                                            ;91's END or EOI bits
```

231324-34

```
106D CA7F10          JZ      SEND5    ;If char = EOS , go finish
             SEND4:  WAITO
1070 DB61    ??0006:  IN     INTl     ;Get Intl status
1072 E602            ANI     BOM      ;Check for byte out
1074 CA7010          JZ      ??0006   ;If not, try again
1077 13              INX     D        ;Increment buffer pointer
1078 0D              DCR     C        ;Decrement count
1079 C26910          JNZ     SEND3    ;If count < > 0, go send
107C C38810          JMP     SEND6    ;Else go finish
107F 13      SEND5:  INX     D        ;for consistency
1080 0D              DCR     C        ; "         "
                     WAITO
                                      ;This ensures that the standard entry
1081 DB61    ??0007:  IN     INTl     ;Get Intl status
1083 E602            ANI     BOM      ;Check for byte out
1085 CA8110          JZ      ??0007   ;If not, try again
                                      ;assumptions for the next subroutine are met
1088 3EFD    SEND6:  MVI     A,TCSY   ;Take control syncronously
108A D369            OUT     CMD92
108C 3E80            MVI     A,AXRA   ;Reset send EOI on EOS
108E D365            OUT     AUXMD
                     WAITX            ;Wait for TCI false
1090 DB6F    ??0008: IN      PRTF
1092 E602            ANI     TCIF
1094 C29010          JNZ     ??0008
                     WAITT            ;Wait for TCI
1097 DB6F    ??0009: IN      PRTF     ;Get task complete int,etc.
1099 E602            ANI     TCIF     ;Mask it
109B CA9710          JZ      ??0009   ;Wait for task to be complete
109E C9              RET
             ;*********************************************************************
             ;
             ;       RECEIVE ROUTINE
             ;
             ;
             ;INPUT:          HL talker pointer
             ;                DE data buffer pointer
             ;                C  count (max buffer size) 0 implies 256
             ;                B  EOS character
             ;OUTPUT:         Fills buffer pointed at by DE
             ;CALLS:          None
             ;DESTROYS:       A, BC, DE, HL, F
             ;
             ;RETURNS:        A=0 normal termination--EOS detected
             ;                A=40 Error--- count overrun
             ;                A<40 or A>5EH Error--- bad talk address
             ;
             ;
109F 78      RECV:   MOV     A,B      ;Get EOS character
10A0 D367            OUT     EOSR     ;Output it to 91
                     RANGE   40H,5EH,RECV6
                                      ;Checks for value in range
                                      ;branches to label if not
                                      ;in range. Falls through if
                                      ;lower <= ( (H)(L) ) <= upper.
                                      ;Get next byte.
10A2 7E              MOV     A,M
10A3 FE40            CPI     40H
10A5 FA3911          JM      RECV6
10A8 FE5F            CPI     5EH+1
10AA F23911          JP      RECV6
                                      ;valid if 40H<= talk <=5EH
10AD D360            OUT     DOUT     ;Output talker to GPIB
10AF 23              INX     H        ;Incr pointer for consistency
                     WAITO
10B0 DB61    ??0010:  IN     INTl     ;Get Intl status
10B2 E602            ANI     BOM      ;Check for byte out
10B4 CAB010          JZ      ??0010   ;If not, try again
10B7 3E3F            MVI     A,UNL    ;Stop other listeners
10B9 D360            OUT     DOUT
                     WAITO
10BB DB61    ??0011:  IN     INTl     ;Get Intl status
10BD E602            ANI     BOM      ;Check for byte out
10BF CABB10          JZ      ??0011   ;If not, try again
```

231324-35

```
10C2 3E21                     MVI     A,MLA      ;For completeness
10C4 D360                     OUT     DOUT
10C6 3E86                     MVI     A,AXRA+HOEND+EDEOS      ;End when
10C8 D365                     OUT     AUXMD      ;EOS or EOI & Holdoff
                              WAITO
10CA DB61           +??0012:  IN      INT1       ;Get Intl status
10CC E602           +         ANI     BOM        ;Check for byte out
10CE CACA10         +         JZ      ??0012     ;If not, try again
10D1 3E40                     MVI     A,LON      ;Listen only
10D3 D364                     OUT     ADRMD
                              CLRA               ;Immediate XEQ PON
10D5 AF             +         XRA     A          ;A XOR A =0
10D6 D365                     OUT     AUXMD
10D8 3EF6                     MVI     A,GTSB     ;Goto standby
10DA D359                     OUT     CMD92
                              WAITX              ;Wait for TCI=0
10DC DB6F           +??0013:  IN      PRTF
10DE E602           +         ANI     TCIF
10E0 C2DC10         +         JNZ     ??0013
                              WAITT              ;Wait for TCI=1
10E3 DB6F           +??0014:  IN      PRTF       ;Get task complete int,etc.
10E5 E602           +         ANI     TCIF       ;Mask it
10EA DB61            RECV1:   IN      INT1       ;Get 91 Int status (END &/or BI)
10EC 47                       MOV     B,A        ;Save it in B for BI check later
10ED E610                     ANI     ENDMK      ;Check for EOS or EOI
10EF C20511                   JNZ     RECV2      ;Yes end--- go wait for BI
10F2 78                       MOV     A,B        ;NO, retrieve status &
10F3 E601                     ANI     BIM        ;check for BI
10F5 CAEA10                   JZ      RECV1      ;NO, go wait for either END or BI
10F8 DB60                     IN      DIN        ;YES, BI--- get data
10FA 12                       STAX    D          ;Store it in buffer
10FB 13                       INX     D          ;Increment buffer pointer
10FC 0D                       DCR     C          ;Decrement counter
10FD C2EA10                   JNZ     RECV1      ;If count < > 0 go back & wait
1100 0640                     MVI     B,40H      ;Else set error indicator
1102 C31711                   JMP     RECV5      ;And go take control
                     ;
1105 78              RECV2:   MOV     A,B        ;Retreive status
1106 E601            RECV3:   ANI     BIM        ;Check for BI
1108 C21011                   JNZ     RECV4      ;If BI then go input data
110B DB61                     IN      INT1       ;Else wait for last BI
110D C30611                   JMP     RECV3      ;In loop
1110 DB60            RECV4:   IN      DIN        ;Get data byte
1112 12                       STAX    D          ;Store it in buffer
1113 13                       INX     D          ;Incr data pointer
1114 0D                       DCR     C          ;Decrement count, but ignore it
1115 0600                     MVI     B,0        ;Set normal completion indicators
                     ;
1117 3EFD            RECV5:   MVI     A,TCSY     ;Take control synchronously
1119 D369                     OUT     CMD92
                              WAITX              ;Wait for TCI=0 (7 tcy)
111B DB6F           +??0015:  IN      PRTF
111D E602           +         ANI     TCIF
111F C21B11         +         JNZ     ??0015
                              WAITT              ;Wait for TCI=1
1122 DB6F           +??0016:  IN      PRTF       ;Get task complete int,etc.
1124 E602           +         ANI     TCIF       ;Mask it
1126 CA2211         +         JZ      ??0016     ;Wait for task to be complete
                     ;
                     ;if timeout 3 is to be checked, the above WAITT should
                     ;be omitted & the appropriate code to look for TCI or
                     ;TOUT3 inserted here.
                     ;
1129 3E80                     MVI     A,AXRA     ;Pattern to clear 91 END conditions
112B D365                     OUT     AUXMD      ;
112D 3E80                     MVI     A,TON      ;This bit pattern already in "A"
112F D364                     OUT     ADRMD      ;Output TON
1131 3E03                     MVI     A,FNHSK    ;Finish handshake
1133 D365                     OUT     AUXMD
                              CLRA
1135 AF             +         XRA     A          ;A XOR A =0
1136 D365                     OUT     AUXMD      ;Immediate execute PON-Reset LON
1138 78                       MOV     A,B        ;Get completion character
1139 C9              RECV6:   RET
```

231324-36

```
                 ;
                 ;******************************************************************
                 ;          XFER ROUTINE
                 ;
                 ;
                 ;INPUTS:            HL device list pointer
                 ;                   B  EOS character
                 ;OUTPUTS:           None
                 ;CALLS:             None
                 ;DESTROYS:          A, HL, F
                 ;RETURNS:           A=0 normal, A < > 0 bad talker
                 ;
                 ;
                 ;NOTE:              XFER will not work if the talker
                 ;                   uses EOI to terminate the transfer.
                 ;                   Intel will be making hardware
                 ;                   modifications to the 8291 that will
                 ;                   correct this problem. Until that time,
                 ;                   only EOS may be used without possible
                 ;                   loss of the last data byte transfered.
                 XFER:     RANGE     40H,5EH,XFER4    ;Check for valid talker
                                              ;Checks for value in range
                                              ;branches to label if not
                                              ;in range. Falls through if
                                              ;lower <= ( (H)(L) ) <= upper.
                                              ;Get next byte.
113A 7E                    MOV       A,M
113B FE40                  CPI       40H
113D FABB11                JM        XFER4
1140 FE5F                  CPI       5EH+1
1142 F2BB11                JP        XFER4
1145 D360                  OUT       DOUT     ;Send it to GPIB
1147 23                    INX       H        ;Incr pointer
                           WAITO
1148 DB61        ??0017:   IN        INT1     ;Get Intl status
114A E602                  ANI       BOM      ;Check for byte out
114C CA4811                JZ        ??0017   ;If not, try again
114F 3E3F                  MVI       A,UNL    ;Universal unlisten
1151 D360                  OUT       DOUT
                 XFER1:    RANGE     20H,3EH,XFER2    ;Check for valid listener
                                              ;Checks for value in range
                                              ;branches to label if not
                                              ;in range. Falls through if
                                              ;lower <= ( (H)(L) ) <= upper.
                                              ;Get next byte.
1153 7E                    MOV       A,M
1154 FE20                  CPI       20H
1156 FA6C11                JM        XFER2
1159 FE3F                  CPI       3EH+1
115B F26C11                JP        XFER2
                           WAITO
115E DB61        ??0018:   IN        INT1     ;Get Intl status
1160 E602                  ANI       BOM      ;Check for byte out
1162 CA5E11                JZ        ??0018   ;If not, try again
1165 7E                    MOV       A,M      ;Get listener
1166 D360                  OUT       DOUT
1168 23                    INX       H        ;Incr pointer
1169 C35311                JMP       XFER1    ;Loop until non-valid listener
                 XFER2:    WAITO
116C DB61        ??0019:   IN        INT1     ;Get Intl status
116E E602                  ANI       BOM      ;Check for byte out
1170 CA6C11                JZ        ??0019   ;If not, try again
1173 3E87                  MVI       A,AXRA+CAHCY+EDEOS      ;Invisible handshake
1175 D365                  OUT       AUXMD    ;Continuous AH mode
1177 3E40                  MVI       A,LON    ;Listen only
1179 D364                  OUT       ADRMD
                           CLRA
117B AF                    XRA       A        ;A XOR A =0
117C D365                  OUT       AUXMD    ;Immed. XEQ PON
117E 78                    MOV       A,B      ;Get EOS
117F D367                  OUT       EOSR     ;Output it to 91
1181 3EF6                  MVI       A,GTSB   ;Go to standby
1183 D369                  OUT       CMD92
```

231324-37

```
                         WAITX
1185 DB6F       ??0020:  IN      PRTF
1187 E602                ANI     TCIF
1189 C28511              JNZ     ??0020
                         WAITT             ;Wait for TCS
118C DB6F       ??0021:  IN      PRTF      ;Get task complete int,etc.
118E E602                ANI     TCIF      ;Mask it
1190 CA8C11              JZ      ??0021    ;Wait for task to be complete
1193 DB61       XFER3:   IN      INT1      ;Get END status bit
1195 E610                ANI     ENDMK     ;Mask it
1197 CA9311              JZ      XFER3
119A 3EFD                MVI     A,TCSY    ;Take control syncronously
119C D369                OUT     CMD92
                         WAITX
119E DB6F       ??0022:  IN      PRTF
11A0 E602                ANI     TCIF
11A2 C29E11              JNZ     ??0022
                         WAITT             ;Wait for TCI
11A5 DB6F       ??0023:  IN      PRTF      ;Get task complete int,etc.
11A7 E602                ANI     TCIF      ;Mask it
11A9 CAA511              JZ      ??0023    ;Wait for task to be complete
11AC 3E80                MVI     A,AXRA    ;Not cont AH or END on EOS
11AE D365                OUT     AUXMD
11B0 3E03                MVI     A,FNHSK   ;Finish handshake
11B2 D365                OUT     AUXMD
11B4 3E80                MVI     A,TON     ;Talk only
11B6 D364                OUT     ADRMD
                         CLRA              ;Normal return A=0
11B8 AF                  XRA     A         ;A XOR A =0
11B9 D365                OUT     AUXMD     ;Immediate XEQ PON
11BB C9         XFER4:   RET
                ;
                ;******************************************************
                ;
                ;
                ;        TRIGGER ROUTINE
                ;
                ;
                ;INPUTS:         HL listener list pointer
                ;OUTPUTS:        None
                ;CALLS:          None
                ;DESTROYS:       A, HL, F
                ;
                ;
11BC 3E3F       TRIG:    MVI     A,UNL     ;
11BE D360                OUT     DOUT      ;Send universal unlisten
                TRIG1:   RANGE   20H,3EH,TRIG2   ;Check for valid listen
                                           ;Checks for value in range
                                           ;branches to label if not
                                           ;in range. Falls through if
                                           ;lower <= ( (H)(L) ) <= upper.
                                           ;Get next byte.
11C0 7E                  MOV     A,M
11C1 FE20                CPI     20H
11C3 FAD911              JM      TRIG2
11C6 FE3F                CPI     3EH+1
11C8 F2D911              JP      TRIG2
                         WAITO             ;Wait for UNL to finish
11CB DB61       ??0024:  IN      INT1      ;Get Intl status
11CD E602                ANI     BOM       ;Check for byte out
11CF CACB11              JZ      ??0024    ;If not, try again
11D2 7E                  MOV     A,M       ;Get listener
11D3 D360                OUT     DOUT      ;Send Listener to GPIB
11D5 23                  INX     H         ;Incr. pointer
11D6 C3C011              JMP     TRIG1     ;Loop until non-valid char
                TRIG2:   WAITO             ;Wait for last listen to finish
11D9 DB61       ??0025:  IN      INT1      ;Get Intl status
11DB E602                ANI     BOM       ;Check for byte out
11DD CAD911              JZ      ??0025    ;If not, try again
11E0 3E08                MVI     A,GET     ;Send group execute trigger
11E2 D360                OUT     DOUT      ;to all addressed listeners
                         WAITO
11E4 DB61       ??0026:  IN      INT1      ;Get Intl status
11E6 E602                ANI     BOM       ;Check for byte out
```

231324-38

```
11E8 CAE411    +          JZ      ??0026  ;If not, try again
11EB C9                   RET
               ;
               ;********************************************
               ;
               ;DEVICE CLEAR ROUTINE
               ;
               ;
               ;
               ;INPUTS:            HL listener pointer
               ;OUTPUT:            None
               ;CALLS:             None
               ;DESTROYS:          A, HL, F
               ;
11EC 3E3F       DCLR:     MVI     A,UNL
11EE D360                 OUT     DOUT
                DCLR1:    RANGE   20H,3EH,DCLR2
               +                          ;Checks for value in range
               +                          ;branches to label if not
               +                          ;in range. Falls through if
               +                          ;lower <= ( (H)(L) ) <= upper.
               +                          ;Get next byte.
11F0 7E        +          MOV     A,M
11F1 FE20      +          CPI     20H
11F3 FA0912    +          JM      DCLR2
11F6 FE3F      +          CPI     3EH+1
11F8 F20912    +          JP      DCLR2
                          WAITO
11FB DB61      +??0027:    IN     INT1    ;Get Intl status
11FD E602      +          ANI     BOM     ;Check for byte out
11FF CAFB11    +          JZ      ??0027  ;If not, try again
1202 7E                   MOV     A,M
1203 D360                 OUT     DOUT    ;Send listener to GPIB
1205 23                   INX     H
1206 C3F011               JMP     DCLR1
                DCLR2:    WAITO
1209 DB61      +??0028:    IN     INT1    ;Get Intl status
120B E602      +          ANI     BOM     ;Check for byte out
120D CA0912    +          JZ      ??0028  ;If not, try again
1210 3E04                 MVI     A,SDC   ;Send device clear
1212 D360                 OUT     DOUT    ;To all addressed listeners
                          WAITO
1214 DB61      +??0029:    IN     INT1    ;Get Intl status
1216 E602      +          ANI     BOM     ;Check for byte out
1218 CA1412    +          JZ      ??0029  ;If not, try again
121B C9                   RET
               ;
               ;****************************************************
               ;
               ;          SERIAL POLL ROUTINE
               ;
               ;INPUTS:            HL talker list pointer
               ;                   DE status buffer pointer
               ;OUTPUTS:           Fills buffer pointed to by DE
               ;CALLS:             None
               ;DESTROYS:          A, BC, DE, HL, F
               ;
121C 3E3F       SPOL:     MVI     A,UNL   ;Universal unlisten
121E D360                 OUT     DOUT
                          WAITO
1220 DB61      +??0030:    IN     INT1    ;Get Intl status
1222 E602      +          ANI     BOM     ;Check for byte out
1224 CA2012    +          JZ      ??0030  ;If not, try again
1227 3E21                 MVI     A,MLA   ;My listen address
1229 D360                 OUT     DOUT
                          WAITO
122B DB61      +??0031:    IN     INT1    ;Get Intl status
122D E602      +          ANI     BOM     ;Check for byte out
122F CA2B12    +          JZ      ??0031  ;If not, try again
1232 3E18                 MVI     A,SPE   ;Serial poll enable
1234 D360                 OUT     DOUT    ;To be formal about it
                          WAITO
1236 DB61      +??0032:    IN     INT1    ;Get Intl status
```

231324-39

```
1238 E602          711+          ANI     BOM      ;Check for byte out
123A CA3512        712+          JZ      ??0032   ;If not, try again
                   713 SPOL1:    RANGE   40H,5EH,SPOL2    ;Check for valid talker
                   714+                           ;Checks for value in range
                   715+                           ;branches to label if not
                   716+                           ;in range. Falls through if
                   717+                           ;lower <= ( (H)(L) ) <= upper.
                   718+                           ;Get next byte.
123D 7E            719+          MOV     A,M
123E FE40          720+          CPI     40H
1240 FA9412        721+          JM      SPOL2
1243 FE5F          722+          CPI     5EH+1
1245 F29412        723+          JP      SPOL2
1248 7E            724           MOV     A,M      ;Get talker
1249 D360          725           OUT     DOUT     ;Send to GPIB
124B 23            726           INX     H        ;Incr talker list pointer
124C 3E40          727           MVI     A,LON    ;Listen only
124E D364          728           OUT     ADRMD
                   729           WAITO            ;Wait for talk address to complete
1250 DB61          730+??0033:   IN      INT1     ;Get Intl status
1252 E602          731+          ANI     BOM      ;Check for byte out
1254 CA5012        732+          JZ      ??0033   ;If not, try again
                   733           CLRA             ;Pattern for immediate XEQ PON
1257 AF            734+          XRA     A        ;A XOR A =0
1258 D365          735           OUT     AUXMD
125A 3EF6          736           MVI     A,GTSB   ;Goto standby
125C D369          737           OUT     CMD92
                   738           WAITX            ;Wait for TCI false
125E DB6F          739+??0034:   IN      PRTF
1260 E602          740+          ANI     TCIF
1262 C25E12        741+          JNZ     ??0034
                   742           WAITT            ;Wait for TCI
1265 DB6F          743+??0035:   IN      PRTF     ;Get task complete int,etc.
1267 E602          744+          ANI     TCIF     ;Mask it
1269 CA6512        745+          JZ      ??0035   ;Wait for task to be complete
                   746           WAITI            ;Wait for status byte input
126C DB61          747+??0036:   IN      INT1     ;Get INT1 status
126E 47            748+          MOV     B,A      ;Save status in B
126F E601          749+          ANI     BIM      ;Check for byte in
1271 CA6C12        750+          JZ      ??0036   ;If not, just try again
1274 3EFD          751           MVI     A,TCSY   ;Take control sync
1276 D369          752           OUT     CMD92
                   753           WAITX            ;Wait for TCI false
1278 DB6F          754+??0037:   IN      PRTF
127A E602          755+          ANI     TCIF
127C C27812        756+          JNZ     ??0037
                   757           WAITT            ;Wait for TCI
127F DB6F          758+??0038:   IN      PRTF     ;Get task complete int,etc.
1281 E602          759+          ANI     TCIF     ;Mask it
1283 CA7F12        760+          JZ      ??0038   ;Wait for task to be complete
1286 DB60          761           IN      DIN      ;Get serial poll status byte
1288 12            762           STAX    D        ;Store it in buffer
1289 13            763           INX     D        ;Incr pointer
128A 3E80          764           MVI     A,TON    ;Talk only for controller
128C D364          765           OUT     ADRMD    ;
                   766           CLRA
128E AF            767+          XRA     A        ;A XOR A =0
128F D365          768           OUT     AUXMD    ;Immeditate XEQ PON
                   769                            ;CLR LA
1291 C33D12        770           JMP     SPOL1    ;Go on to next device on list
                   771 ;
1294 3E19          772 SPOL2:    MVI     A,SPD    ;Serial poll disable
1296 D360          773           OUT     DOUT     ;We know BO was set (WAITO above)
                   774           WAITO
1298 DB61          775+??0039:   IN      INT1     ;Get Intl status
129A E602          776+          ANI     BOM      ;Check for byte out
129C CA9812        777+          JZ      ??0039   ;If not, try again
                   778           CLRA
129F AF            779+          XRA     A        ;A XOR A =0
12A0 D365          780           OUT     AUXMD    ;Immediate XEQ PON to clear LA
12A2 C9            781           RET
                   782 ;
                   783 ;*****************************************************
                   784 ;
```

231324-40

```
                 ;           PARALLEL POLL ENABLE ROUTINE
                 ;
                 ;INPUTS:              HL listener list pointer
                 ;                     DE configuration byte pointer
                 ;OUTPUTS:             None
                 ;CALLS:               None
                 ;DESTROYS:            A, DE, HL, F
                 ;
                 ;
12A3 3E3F        PPEN:     MVI         A,UNL       ;Universal unlisten
12A5 D360                  OUT         DOUT
                 PPEN1:    RANGE       20H,3EH,PPEN2     ;Check for valid listener
                                                   ;Checks for value in range
                                                   ;branches to label if not
                                                   ;in range. Falls through if
                                                   ;lower <= ( (H)(L) ) <= upper.
                                                   ;Get next byte.
12A7 7E                    MOV         A,M
12A8 FE20                  CPI         20H
12AA FAD812                JM          PPEN2
12AD FE3F                  CPI         3EH+1
12AF F2D812                JP          PPEN2
                           WAITO                   ;Valid wait 91 data out reg
12B2 DB61        ??0040:   IN          INT1        ;Get Intl status
12B4 E602                  ANI         BOM         ;Check for byte out
12B6 CAB212                JZ          ??0040      ;If not, try again
12B9 7E                    MOV         A,M         ;Get listener
12BA D360                  OUT         DOUT
                           WAITO
12BC DB61        ??0041:   IN          INT1        ;Get Intl status
12BE E602                  ANI         BOM         ;Check for byte out
12C0 CABC12                JZ          ??0041      ;If not, try again
12C3 3E05                  MVI         A,PPC       ;Parallel poll configure
12C5 D360                  OUT         DOUT
                           WAITO
12C7 DB61        ??0042:   IN          INT1        ;Get Intl status
12C9 E602                  ANI         BOM         ;Check for byte out
12CB CAC712                JZ          ??0042      ;If not, try again
12CE 1A                    LDAX        D           ;Get matching configuration byte
12CF F660                  ORI         PPE         ;Merge with parallel poll enable
12D1 D360                  OUT         DOUT
12D3 23                    INX         H           ;Incr pointers
12D4 13                    INX         D
12D5 C3A712                JMP         PPEN1       ;Loop until invalid listener char
                 PPEN2:    WAITO
12D8 DB61        ??0043:   IN          INT1        ;Get Intl status
12DA E602                  ANI         BOM         ;Check for byte out
12DC CAD812                JZ          ??0043      ;If not, try again
12DF C9                    RET
                 ;
                 ;PARALLEL POLL DISABLE ROUTINE
                 ;
                 ;INPUTS:              HL listener list pointer
                 ;OUTPUTS:             None
                 ;CALLS:               None
                 ;DESTROYS:            A, HL, F
                 ;
12E0 3E3F        PPDS:     MVI         A,UNL       ;Universal unlisten
12E2 D360                  OUT         DOUT
                 PPDS1:    RANGE       20H,3EH,PPDS2     ;Check for valid listener
                                                   ;Checks for value in range
                                                   ;branches to label if not
                                                   ;in range. Falls through if
                                                   ;lower <= ( (H)(L) ) <= upper.
                                                   ;Get next byte.
12E4 7E                    MOV         A,M
12E5 FE20                  CPI         20H
12E7 FAFD12                JM          PPDS2
12EA FE3F                  CPI         3EH+1
12EC F2FD12                JP          PPDS2
                           WAITO
12EF DB61        ??0044:   IN          INT1        ;Get Intl status
12F1 E602                  ANI         BOM         ;Check for byte out
12F3 CAEF12                JZ          ??0044      ;If not, try again
```

231324-41

```
12F6 7E                      MOV     A,M      ;Get listener
12F7 D360                    OUT     DOUT
12F9 23                      INX     H        ;Incr pointer
12FA C3E412                  JMP     PPDS1    ;Loop until invalid listener
           PPDS2:            WAITO
12FD DB61       ??0045:      IN      INT1     ;Get Intl status
12FF E602                    ANI     BOM      ;Check for byte out
1301 CAFD12                  JZ      ??0045   ;If not, try again
1304 3E05                    MVI     A,PPC    ;Parallel poll configure
1306 D360                    OUT     DOUT
                             WAITO
1308 DB61       ??0046:      IN      INT1     ;Get Intl status
130A E602                    ANI     BOM      ;Check for byte out
130C CA0813                  JZ      ??0046   ;If not, try again
130F 3E70                    MVI     A,PPD    ;Parallel poll disable
1311 D360                    OUT     DOUT
                             WAITO
1313 DB61       ??0047:      IN      INT1     ;Get Intl status
1315 E602                    ANI     BOM      ;Check for byte out
1317 CA1313                  JZ      ??0047   ;If not, try again
131A C9                      RET
           ;
           ;                 PARALLEL POLL UNCONFIGURE ALL ROUTINE
           ;
           ;
           ;INPUTS:                  None
           ;OUTPUTS:                 None
           ;CALLS:                   None
           ;DESTROYS:                A, F
           ;
131B 3E15  PPUN:             MVI     A,PPU    ;Parallel poll unconfigure
131D D360                    OUT     DOUT
                             WAITO
131F DB61       ??0048:      IN      INT1     ;Get Intl status
1321 E602                    ANI     BOM      ;Check for byte out
1323 CA1F13                  JZ      ??0048   ;If not, try again
1326 C9                      RET
           ;
           ;****************************************************
           ;
           ;CONDUCT A PARALLEL POLL
           ;
           ;
           ;INPUTS:                  None
           ;OUTPUTS:                 None
           ;CALLS:                   None
           ;DESTROYS:                A, B, F
           ;RETURNS:                 A= parallel poll status byte
           ;
1327 3E40  PPOL:             MVI     A,LON    ;Listen only
1329 D364                    OUT     ADRMD
                             CLRA             ;Immediate XEQ PON
132B AF                      XRA     A        ;A XOR A =0
132C D365                    OUT     AUXMD    ;Reset TON
132E 3EF5                    MVI     A,EXPP   ;Execute parallel poll
1330 D369                    OUT     CMD92
                             WAITI            ;Wait for completion= BI on 91
1332 DB61       ??0049:      IN      INT1     ;Get INT1 status
1334 47                      MOV     B,A      ;Save status in B
1335 E601                    ANI     BIM      ;Check for byte in
1337 CA3213                  JZ      ??0049   ;If not, just try again
133A 3E80                    MVI     A,TON    ;Talk only
133C D364                    OUT     ADRMD
                             CLRA             ;Immediate XEQ PON
133E AF                      XRA     A        ;A XOR A =0
133F D365                    OUT     AUXMD    ;Reset LON
1341 DB60                    IN      DIN      ;Get PP byte
1343 C9                      RET
           ;
           ;****************************************************
           ;PASS CONTROL ROUTINE
           ;
           ;INPUTS:                  HL pointer to talker
           ;OUTPUTS:                 None
```

231324-42

```
                  ;CALLS:           None
                  ;DESTROYS:        A, HL, F
                  PCTL:     RANGE   40H,5EH,PCTL1   ;Is it a valid talker ?
                                            ;Checks for value in range
                                            ;branches to label if not
                                            ;in range. Falls through if
                                            ;lower <= ( (H)(L) ) <= upper.
                                            ;Get next byte.
1344 7E                     MOV     A,M
1345 FE40                   CPI     40H
1347 FA8A13                 JM      PCTL1
134A FE5F                   CPI     5EH+1
134C F28A13                 JP      PCTL1
134F FE41                   CPI     MTA     ;Is it my talker address
1351 CA8A13                 JZ      PCTL1   ;Yes, just return
1354 D360                   OUT     DOUT    ;Send on GPIB
                            WAITO
1356 DB61         ??0050:   IN      INT1    ;Get Intl status
1358 E602                   ANI     BOM     ;Check for byte out
135A CA5613                 JZ      ??0050  ;If not, try again
135D 3E09                   MVI     A,TCT   ;Take control message
135F D360                   OUT     DOUT
                            WAITO
1361 DB61         ??0051:   IN      INT1    ;Get Intl status
1363 E602                   ANI     BOM     ;Check for byte out
1365 CA6113                 JZ      ??0051  ;If not, try again
1368 3E01                   MVI     A,MODE1 ;Not talk only or listen only
136A D364                   OUT     ADRMD   ;Enable 91 address mode 1
                            CLRA
136C AF                     XRA     A       ;A XOR A =0
136D D365                   OUT     AUXMD   ;Immediate XEQ PON
136F 3E01                   MVI     A,MDA   ;My device address
1371 D366                   OUT     ADR01   ;enabled to talk and listen
1373 3EA1                   MVI     A,AXRB+CPTEN    ;Command pass thru enable
1375 D365                   OUT     AUXMD
                  ;*******optional PP configuration goes here********
1377 3EF1                   MVI     A,GIDL  ;92 go idle command
1379 D369                   OUT     CMD92
                            WAITX
137B DB6F         ??0052:   IN      PRTF
137D E602                   ANI     TCIF
137F C27B13                 JNZ     ??0052
                            WAITT           ;Wait for TCI
1382 DB6F         ??0053:   IN      PRTF    ;Get task complete int,etc.
1384 E602                   ANI     TCIF    ;Mask it
1385 CA8213                 JZ      ??0053  ;Wait for task to be complete
1389 23                     INX     H
138A C9           PCTL1:    RET
                  ;
                  ;
                  ;*********************************************
                  ;
                  ;RECEIVE CONTROL ROUTINE
                  ;
                  ;INPUTS:          None
                  ;OUTPUTS:         None
                  ;CALLS:           None
                  ;DESTROYS:        A, F
                  ;RETURNS:         0= invalid (not take control to us or CPT bit not on)
                  ;                 < > 0 = valid take control-- 92 will now be in control
                  ;NOTE:            THIS CODE MUST BE TIGHTLY INTEGRATED INTO ANY USER
                  ;                 SOFTWARE THAT FUNCTIONS WITH THE 8291 AS  A DEVICE.
                  ;                 NORMALLY SOME ADVANCE WARNING OF IMPENDING PASS
                  ;                 CONTROL SHOULD BE GIVEN TO US BY THE CONTROLLER
                  ;                 WITH OTHER USEFUL INFO. THIS PROTOCOL IS SITUATION
                  ;                 SPECIFIC AND WILL NOT BE COVERED HERE.
                  ;
                  ;
138B DB61         RCTL:     IN      INT1    ;Get INT1 req (i.e. CPT etc.)
138D E680                   ANI     CPT     ;Is command pass thru on ?
138F CACF13                 JZ      RCTL2   ;No, invalid-- go return
1392 DB65                   IN      CPTRG   ;Get command
1394 FE09                   CPI     TCT     ;Is it take control ?
```

231324-43

```
1396 C2CA13             JNZ     RCTL1    ;No, go return invalid
1399 DB64               IN      ADRST    ;Get address status
139B E602               ANI     TA       ;Is TA on ?
139D CACA13             JZ      RCTL1    ;No -- go return invalid
13A0 3E60               MVI     A,DTDL1  ;Disable talker listener
13A2 D366               OUT     ADR01
13A4 3E80               MVI     A,TON    ;Talk only
13A6 D364               OUT     ADRMD
                        CLRA
13A8 AF                 XRA     A        ;A XOR A =0
13A9 D361               OUT     INT1     ;Mask off INT bits
13AB D362               OUT     INT2
13AD D365               OUT     AUXMD
13AF 3EFA               MVI     A,TCNTR  ;Take (receive) control 92 command
13B1 D369               OUT     CMD92
13B3 3E0F               MVI     A,VSCMD  ;Valid command pattern for 91
13B5 D365               OUT     AUXMD
             ;******** optional TOUT1 check could be put here ********
                        WAITX
13B7 DB6F    ??0054:    IN      PRTF
13B9 E602               ANI     TCIF
13BB C2B713             JNZ     ??0054
                        WAITT            ;Wait for TCI
13BE DB6F    ??0055:    IN      PRTF     ;Get task complete int,etc.
13C0 E602               ANI     TCIF     ;Mask it
13C2 CABE13             JZ      ??0055   ;Wait for task to be complete
13C5 3E09               MVI     A,TCT    ;Valid return pattern
13C7 C3CF13             JMP     RCTL2    ;Only one return per routine
13CA 3E0F    RCTL1:     MVI     A,VSCMD  ;Acknowledge CPT
13CC D365               OUT     AUXMD
                        CLRA             ;Error return pattern
13CE AF                 XRA     A        ;A XOR A =0
13CF C9      RCTL2:     RET
             ;
             ;*******************************************************
             ;
             ;          SRQ ROUTINE
             ;
             ;INPUTS:            None
             ;OUTPUTS:           None
             ;CALLS:             None
             ;RETURNS:           A= 0 no SRQ
             ;                   A < > 0 SRQ occured
             ;
             ;
13D0 DB69    SRQD:      IN      INTST    ;Get 92's INTRQ status
13D2 E620               ANI     SRQBT    ;Mask off SRQ
13D4 CAE213             JZ      SRQD2    ;Not set--- go return
13D7 F60B               ORI     IACK     ;Set--- must clear it with IACK
13D9 D369               OUT     CMD92
13DB DB69    SRQD1:     IN      INTST    ;Get IBF
13DD E602               ANI     IBFBT    ;Mask it
13DF CADB13             JZ      SRQD1    ;Wait if not set
13E2 C9      SRQD2:     RET
             ;
             ;***********************************************
             ;
             ;REMOTE ENABLE ROUTINE
             ;
             ;INPUTS:            None
             ;OUTPUTS:           None
             ;CALLS:             NONE
             ;DESTROYS:          A, F
             ;
13E3 3EF8    REME:      MVI     A,SREM
13E5 D369               OUT     CMD92    ;92 asserts remote enable
                        WAITX            ;Wait for TCI = 0
13E7 DB6F    ??0056:    IN      PRTF
13E9 E602               ANI     TCIF
13EB C2E713             JNZ     ??0056
                        WAITT            ;Wait for TCI
13EE DB6F    ??0057:    IN      PRTF     ;Get task complete int,etc.
13F0 E602               ANI     TCIF     ;Mask it
13F2 CAEE13             JZ      ??0057   ;Wait for task to be complete
```

231324-44

```
13F5 C9                      RET
                  ;
                  ;**********************************************
                  ;
                  ;LOCAL ROUTINE
                  ;
                  ;
                  ;INPUTS:                None
                  ;OUTPUTS:               None
                  ;CALLS:                 None
                  ;DESTROYS:              A, F
                  ;
13F6 3EF7         LOCL:      MVI        A,SLOC
13F8 D369                    OUT        CMD92     ;92 stops asserting remote enable
                             WAITX                ;Wait for TCI =0
13FA DB6F         +??0058:   IN         PRTF
13FC E602         +          ANI        TCIF
13FE C2FA13       +          JNZ        ??0058
                             WAITT                ;Wait for TCI
1401 DB6F         +??0059:   IN         PRTF      ;Get task complete int,etc.
1403 E602         +          ANI        TCIF      ;Mask it
1405 CA0114       +          JZ         ??0059    ;Wait for task to be complete
1408 C9                      RET
                  ;
                  ;************************************************
                  ;
                  ;INTERFACE CLEAR / ABORT ROUTINE
                  ;
                  ;
                  ;INPUTS:                          None
                  ;OUTPUTS:                         None
                  ;CALLS:                           None
                  ;DESTROYS:                        A, F
                  ;
                  ;
1409 3EF9         IFCL:      MVI        A,ABORT
140B D369                    OUT        CMD92     ;Send IFC
                             WAITX                ;Wait for TCI =0
140D DB6F         +??0060:   IN         PRTF
140F E602         +          ANI        TCIF
1411 C20D14       +          JNZ        ??0060
                             WAITT                ;Wait for TCI
1414 DB6F         +??0061:   IN         PRTF      ;Get task complete int,etc.
1416 E602         +          ANI        TCIF      ;Mask it
1418 CA1414       +          JZ         ??0061    ;Wait for task to be complete
                  ;Delete both WAITX & WAITT if this routine
                  ;is to be called while the 8292 is
                  ;Controller-in-Charge. If not C.I.C. then
                  ;TCI is set, else nothing is set (IFC is sent)
                  ;and the WAIT'S will hang forever
141B C9                      RET
                  ;
```

231324-45

```
                   ;APPLICATION EXAMPLE CODE FOR 8085
                   ;
0032               FGDNL     EQU     '2'       ;Func gen device num "2" ASCII,lstn
0031               FCDNL     EQU     '1'       ;Freq ctr device num "1" ASCII,lstn
0051               FCDNT     EQU     'Q'       ;Freq ctr talk address
000D               CR        EQU     0DH       ;ASCII carriage return
000A               LF        EQU     0AH       ;ASCII line feed
00FF               LEND      EQU     0FFH      ;List end for Talk/Listen lists
0040               SRQM      EQU     40H       ;Bit indicating device sent SRQ
                   ;
141C 46553146      FGDATA:   DB      'FU1FR37KHAM2VO',CR      ;Data to set up func. gen
1420 5233374B
1424 48414D32
1428 564F
142A 0D
000F               LIM1      EQU     15                       ;Buffer length
142B 50463447      FCDATA:   DB      'PF4G7T'                 ;Data to set up freq ctr
142F 3754
0006               LIM2      EQU     6                        ;Buffer length
1431 31            LL1:      DB      FCDNL,LEND               ;Listen list for freq ctr
1432 FF
1433 32            LL2:      DB      FGDNL,LEND               ;Listen list for func. gen
1434 FF
1435 51            TL1:      DB      FCDNT,LEND               ;Talk list for freq ctr
1436 FF
                   ;
                   ;SETUP FUNCTION GENERATOR
1437 060D                    MVI     B,CR      ;EOS
1439 0E0F                    MVI     C,LIM1    ;Count
143B 111C14                  LXI     D,FGDATA            ;Data pointer
143E 213314                  LXI     H,LL2     ;Listen list pointer
1441 CD1C10                  CALL    SEND
                   ;
                   ;SETUP FREQ COUNTER
                   ;
1444 0654                    MVI     B,'T'     ;EOS
1446 0E06                    MVI     C,LIM2    ;Count
1448 112B14                  LXI     D,FCDATA            ;Data pointer
144B 213114                  LXI     H,LL1     ;Listen list pointer
144E CD1C10                  CALL    SEND
                   ;
                   ;WAIT FOR SRQ FROM FREQ CTR
                   ;
1451 CDD013        LOOP:     CALL    SRQD      ;Has SRQ occurred ?
1454 CA5114                  JZ      LOOP      ;No, wait for it
                   ;
                   ;SERIAL POLL TO CLEAR SRQ
                   ;
1457 11003C                  LXI     D,SPBYTE            ;Buffer pointer
145A 213514                  LXI     H,TL1               ;Talk list pointer
145D CD1C12                  CALL    SPOL
1460 1B                      DCX     D         ;Backup buffer pointer to ctr byte
1461 1A                      LDAX    D         ;Get status byte
1462 E640                    ANI     SRQM      ;Did ctr assert SRQ ?
1464 CA7714                  JZ      ERROR     ;Ctr should have said yes
                   ;
                   ;RECEIVE READING FROM COUNTER
                   ;
1467 060A                    MVI     B,LF      ;EOS
1469 0E11                    MVI     C,LIM3    ;Count
146B 213514                  LXI     H,TL1     ;Talk list pointer
146E 11013C                  LXI     D,FCDATI            ;Data in buffer pointer
1471 CD9F10                  CALL    RECV
1474 C27714                  JNZ     ERROR
                   ;
                   ;******* rest of user processing goes here *****
                   ;
                   ;
1477 00            ERROR:    NOP                 ;User dependant error handling
                   ;         ETC.
3C00               ORG       3C00H
3C00               SPBYTE:   DS      1         ;Location for serial poll byte
0011               LIM3      EQU     17        ;Max freq counter input
```

231324-46

```
3C01              1198 FCDATI: DS       LIM3     ;Freq ctr input buffer
                  1199         END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
ABORT  A 00F9  ADR01  A 0056  ADRMD  A 0064  ADRST  A 0064  AUXMD  A 0065  AXRA   A 0080  AXRB   A 00A0
BIM    A 0001  BOF    A 0001  BOM    A 0002  BUSST  A 0058  CAHCY  A 0003  CLKRT  A 0023  CLRA   + 0007
CLRST  A 0068  CMD92  A 0069  CPT    A 0080  CPTEN  A 0001  CPTRG  A 0055  CR     A 000D  DCL    A 0014
DCLR   A 11EC  DCLR1  A 11F0  DCLR2  A 1209  DIN    A 0060  DOUT   A 0050  DTDL1  A 0050  DTDL2  A 00E0
EDEOS  A 0004  ENDMK  A 0010  EOIS   A 0008  EOIST  A 0020  EOSR   A 0057  ERFLG  A 0068  ERRM   A 0068
ERROR  A 1477  EVBIT  A 0010  EVCST  A 0068  EVREG  A 0068  EXPP   A 00F5  FCDATA A 1428  FCDATI A 3C01
FCDNL  A 0031  FCDNT  A 0051  FGDATA A 141C  FGDNL  A 0032  FNHSK  A 0003  GET    A 0008  GIDL   A 00F1
GSEC   A 00F4  GTSB   A 00F6  HOEND  A 0002  HOHSK  A 0001  IACK   A 003B  IBFBT  A 0002  IBFF   A 0010
IFCL   A 1409  INIT   A 1000  INT1   A 0061  INT2   A 0052  INTM   A 00A0  INTM1  A 0051  INTMR  A 0068
INTST  A 0069  LA     A 0001  LEND   A 00FF  LF     A 000A  LIM1   A 000F  LIM2   A 0005  LIM3   A 0011
LL1    A 1431  LL2    A 1433  LOCL   A 13F6  LON    A 0040  LOOP   A 1451  MDA    A 0001  MLA    A 0021
MODE1  A 0001  MTA    A 0041  NVCMD  A 0007  OBFF   A 0008  PCTL   A 1344  PCTL1  A 138A  PPC    A 0005
PPD    A 0070  PPDS   A 12E0  PPDS1  A 12E4  PPDS2  A 12FD  PPE    A 0060  PPEN   A 12A3  PPEN1  A 12A7
PPEN2  A 12D8  PPOL   A 1327  PPU    A 0015  PPUN   A 131B  PRT91  A 0050  PRT92  A 0068  PRTF   A 005F
RANGE  + 0005  RBST   A 00E7  RCST   A 00E6  RCTL   A 138B  RCTL1  A 13CA  RCTL2  A 13CF  RECV   A 109F
RECV1  A 10EA  RECV2  A 1105  RECV3  A 1106  RECV4  A 1110  RECV5  A 1117  RECV6  A 1139  REME   A 13E3
RERF   A 00E4  RERM   A 00EA  REVC   A 00E3  RINM   A 00E5  RSET   A 00F2  RSTI   A 00F3  RTOUT  A 00E9
SDEOI  A 0006  SEND   A 101C  SEND1  A 102E  SEND2  A 1047  SEND3  A 1059  SEND4  A 1070  SEND5  A 107F
SEND6  A 1088  SETF   + 0003  SLOC   A 00F7  SPBYTE A 3C00  SPCNI  A 00F0  SPD    A 0019  SPE    A 0018
SPIF   A 0004  SPOL   A 121C  SPOL1  A 123D  SPOL2  A 1294  SREM   A 00F8  SROBT  A 0020  SROD   A 13D0
SRQD1  A 13DB  SRQD2  A 13E2  SROM   A 0040  STCNI  A 00FE  TA     A 0002  TCASY  A 00FC  TCIF   A 0002
TCNTR  A 00FA  TCSY   A 00FD  TCT    A 0009  TL1    A 1435  TLON   A 00C0  TON    A 0090  TOREG  A 0068
TOST   A 0068  TOUT1  A 0001  TOUT2  A 0002  TOUT3  A 0004  TRIG   A 11BC  TRIG1  A 11C0  TRIG2  A 11D9
UNL    A 003F  VSCMD  A 000F  WAITI  + 0002  WAITO  + 0001  WAITT  + 0004  WAITX  + 0003  WEVC   A 00E2
WOUT   A 0021  XFER   A 113A  XFER1  A 1153  XFER2  A 116C  XFER3  A 1193  XFER4  A 11BB


ASSEMBLY COMPLETE,    NO ERRORS
```

231324-47

APPENDIX B

Test Cases for the Software Drivers

The following test cases were used to exercise the software routines and to check their action. To provide another device/controller on the GPIB a ZT488 GPIB Analyzer was used. This analyzer acted as a talker, listener or another controller as needed to execute the tests. The sequence of outputs are shown with each test. All numbers are hexadecimal.

Send Test Cases

B=	44	44	44
C=	30	2	0
DE=	3E80	3E80	3E80
HL=	3E70	3E70	3E70
3E70:	20 30 3E 3F		
3E80:	11 44		
GPIB output:	41 ATN	41 ATN	41 ATN
	3F ATN	3F ATN	3F ATN
	20 ATN	20 ATN	20 ATN
	30 ATN	30 ATN	30 ATN
	3E ATN	3E ATN	3E ATN
	11	11	
	44 EOI	44 EOI	
Ending B=	44	44	44
Ending C=	2E	0	0
Ending DE=	3E82	3E82	3E80
Ending HL=	3E73	3E73	3E73

Receive Test Cases

B=	44	44	44	44	44	44	44
C=	30	30	30	30	4	4	0=256
DE=	3E80	3E80	3E80	3E80	3E80	3E80	3E80
HL=	3E70	3E70	3E70	3E70	3E70	3E70	3E70
3E70:	40	50	5E	5F	40	40	40
GPIB output:	40 ATN	50 ATN	5E ATN		40 ATN	40 ATN	40 ATN
	3F ATN	3F ATN	3F ATN		3F ATN	3F ATN	3F ATN
	21 ATN	21 ATN	21 ATN		21 ATN	21 ATN	21 ATN
ZT488 Data	1	1	1		1	11	1
In	2	2	2		2	22	2
	3	3	3		3	33	3
	4	4	44,EOI		4	44	44
	44	5,EOI					
Ending A =	0	0	0	5F	40	0	0
Ending B =	0	0	0	44	40	0	0
Ending C =	2B	2B	2C	30	0	0	FC
Ending DE=	3E85	3E85	3E84	3E80	3E84	3E84	3E84
Ending HL=	3E71	3E71	3E71	3E70	3E71	3E71	3E71

Serial Poll Test Cases

C=	30
DE=	3E80
HL=	3E70
3E70:	40
	50
	5E
	5F
GPIB output:	3F ATN
output:	21 ATN
output:	18 ATN
output:	40 ATN
input*:	00
output:	50 ATN
input*:	41
output:	5E ATN
input*:	7F
output:	19 ATN

*NOTE: leave ZT488 in single step mode even on input

Ending C =	30
Ending DE=	3E83
Ending HL=	3E73
Ending 3E80:	00 41 7F

C=	30
DE=	3E80
HL=	3E70
3E70:	5F
GPIB output:	3F ATN
	21 ATN
	18 ATN
	19 ATN
Ending C =	30
Ending DE=	3E80
Ending HL=	3E70

Pass Control Test Cases

HL=	3E70	3E70	3E70
3E70:	40	41(MTA)	5F
GPIB output:	40 ATN		
	09 ATN		
	—$\overline{\text{ATN}}$		
Ending HL=	3E71	3E70	3E70
Ending A =	02	41(MTA)	5F

Receive Control Test Cases

GPIB input	10 ATN	40 ATN	41 ATN
	$\overline{\text{ATN}}$	09 ATN	09 ATN
Run Receive Control			
GPIB Input		$\overline{\text{ATN}}$	$\overline{\text{ATN}}$
Ending A=	0	0	09

Parallel Poll Enable Test Cases

DE=	3E80	3E80
HL=	3E70	3E70
3E70:	20 30 3E 3F	3F
3E80:	01 02 03	
GPIB output:	3F ATN	3F ATN
	20 ATN	
	05 ATN	
	61 ATN	
	30 ATN	
	05 ATN	
	62 ATN	
	3E ATN	
	05 ATN	
	63 ATN	
Ending DE=	3E83	3E80
Ending HL=	3E73	3E70

Parallel Poll Disable Test Cases

HL=	3E70	3E70
3E70:	20 30 3E 3F	3F
GPIB output:	3F ATN	3F ATN
	20 ATN	05 ATN
	30 ATN	70 ATN
	3E ATN	
	05 ATN	
	70 ATN	
Ending HL=	3E73	3E70

Parallel Poll Unconfigure Test Case

GPIB output: 15 ATN

Parallel Poll Test Cases

Set DIO#	1	2	3	4	5	6	7	8	None
Ending A	1	2	4	8	10	20	40	80	0

SRQ Test

	Set SRQ momentarily	Reset SRQ
Ending A =	02	00

Trigger Test

HL=	3E70
DE=	3E80
BC=	4430
3E70:	20 30 3E 3F
GPIB output:	3F ATN
	20 ATN
	30 ATN
	3E ATN
	08 ATN
Ending HL=	3E73
DE=	3E80
BC=	4430

Device Clear Test

HL=	3E70
DE=	3E80
BC=	4430
3E70:	20 30 3E 3F
GPIB output:	3F ATN
	20 ATN
	30 ATN
	3E ATN
	14 ATN
Ending HL=	3E73
DE=	3E80
RC=	4430

XFER Test

B=	44
HL=	3E70:
3E70:	40 20 30 3E 3F
GPIB output:	40 ATN
	3F ATN
	20 ATN
	30 ATN
	3E ATN
GPIB input:	0
	1
	2
	3
	44
Ending A =	0
B=	44
HL =	3E74

Application Example
GPIB Output/Input

GPIB output:	41 ATN
	3F ATN
	32 ATN
	46
	55
	31
	46
	52
	33
	37
	4B
	48
	41
	4D
	32
	56
	4F
	0D EOI
	41 ATN
	3F ATN
	31 ATN
	50
	46
	34
	47
	37
	54 EOI
GPIB input:	SRQ
GPIB output:	3F ATN
	21 ATN
	18 ATN
	51 ATN
GPIB input:	40 $\overline{SRQ}$
GPIB output:	19 ATN
	51 ATN
	3F ATN
	21 ATN

GPIB input:	20
	2B
	20
	20
	20
	33
	37
	30
	30
	30
	2E
	30
	45
	2B
	30
	0D
	0A
GPIB output:	XX ATN

APPENDIX C

REMOTE MESSAGE CODING

					Bus Signal Line(s) and Coding That Asserts the True Value of the Message													
Mnemonic	Message Name		Type	Class	DIO 8	7	6	5	4	3	2	DIO 1	DAV NRFD NDAC	ATN	EOI	SRQ	IFC	REN
ACG	addressed command group		M	AC	Y	0	0	0	X	X	X	X	XXX	1	X	X	X	X
ATN	attention		U	UC	X	X	X	X	X	X	X	X	XXX	1	X	X	X	X
DAB	data byte	(Notes 1, 9)	M	DD	D	D	D	D	D	D	D	D	XXX	0	X	X	X	X
					8	7	6	5	4	3	2	1						
DAC	data accepted		U	HS	X	X	X	X	X	X	X	X	XX0	X	X	X	X	X
DAV	data valid		U	HS	X	X	X	X	X	X	X	X	1XX	X	X	X	X	X
DCL	device clear		M	UC	Y	0	0	1	0	1	0	0	XXX	1	X	X	X	X
END	end		U	ST	X	X	X	X	X	X	X	X	XXX	0	1	X	X	X
EOS	end of string	(Notes 2, 9)	M	DD	E	E	E	E	E	E	E	E	XXX	0	X	X	X	X
					8	7	6	5	4	3	2	1						
GET	group execute trigger		M	AC	Y	0	0	0	1	0	0	0	XXX	1	X	X	X	X
GTL	go to local		M	AC	Y	0	0	0	0	0	0	1	XXX	1	X	X	X	X
IDY	identify		U	UC	X	X	X	X	X	X	X	X	XXX	X	1	X	X	X
IFC	interface clear		U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	1	X
LAG	listen address group		M	AD	Y	0	1	X	X	X	X	X	XXX	1	X	X	X	X
LLO	local lock out		M	UC	Y	0	0	1	0	0	0	1	XXX	1	X	X	X	X
MLA	my listen address	(Note 3)	M	AD	Y	0	1	L	L	L	L	L	XXX	1	X	X	X	X
								5	4	3	2	1						
MTA	my talk address	(Note 4)	M	AD	Y	1	0	T	T	T	T	T	XXX	1	X	X	X	X
								5	4	3	2	1						
MSA	my secondary address	(Note 5)	M	SE	Y	1	1	S	S	S	S	S	XXX	1	X	X	X	X
								5	4	3	2	1						
NUL	null byte		M	DD	0	0	0	0	0	0	0	0	XXX	X	X	X	X	X
OSA	other secondary address		M	SE	(OSA = SCG $\wedge$ $\overline{\text{MSA}}$)													
OTA	other talk address		M	AD	(OTA = TAG $\wedge$ $\overline{\text{MTA}}$)													
PCG	primary command group		M	—	(PCG = ACG $\vee$ UCG $\vee$ LAG $\vee$ TAG)													
PPC	parallel poll configure		M	AC	Y	0	0	0	0	1	0	1	XXX	1	X	X	X	X
PPE	parallel poll enable	(Note 6)	M	SE	Y	1	1	0	S	P	P	P	XXX	1	X	X	X	X
										3	2	1						
PPD	parallel poll disable	(Note 7)	M	SE	Y	1	1	1	D	D	D	D	XXX	1	X	X	X	X
									4	3	2	1						
PPR1	parallel poll response 1	(Note 10)	U	ST	X	X	X	X	X	X	X	1	XXX	1	1	X	X	X
PPR2	parallel poll response 2	(Note 10)	U	ST	X	X	X	X	X	X	1	X	XXX	1	1	X	X	X

REMOTE MESSAGE CODING (Continued)

Mnemonic	Message Name		Type	Class	DIO 8	7	6	5	4	3	2	DIO 1	DAV NRFD NDAC	ATN	EOI	SRQ	IFC	REN
					Bus Signal Line(s) and Coding That Asserts the True Value of the Message													
PPR3	parallel poll response 3		U	ST	X	X	X	X	X	1	X	X	XXX	1	1	X	X	X
PPR4	parallel poll response 4	(Note 10)	U	ST	X	X	X	X	1	X	X	X	XXX	1	1	X	X	X
PPR5	parallel poll response 5		U	ST	X	X	X	1	X	X	X	X	XXX	1	1	X	X	X
PPR6	parallel poll response 6		U	ST	X	X	1	X	X	X	X	X	XXX	1	1	X	X	X
PPR7	parallel poll response 7	(Note 10)	U	ST	X	1	X	X	X	X	X	X	XXX	1	1	X	X	X
PPR8	parallel poll response 8		U	ST	1	X	X	X	X	X	X	X	XXX	1	1	X	X	X
PPU	parallel poll unconfigure		M	UC	Y	0	0	1	0	1	0	1	XXX	1	X	X	X	X
REN	remote enable		U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	X	1
RFD	ready for data		U	HS	X	X	X	X	X	X	X	X	X0X	X	X	X	X	X
RQS	request service	(Note 9)	U	ST	X	1	X	X	X	X	X	X	XXX	0	X	X	X	X
SCG	secondary command group		M	SE	Y	1	1	X	X	X	X	X	XXX	1	X	X	X	X
SDC	selected device clear		M	AC	Y	0	0	0	0	1	0	0	XXX	1	X	X	X	X
SPD	serial poll disable		M	UC	Y	0	0	1	1	0	0	1	XXX	1	X	X	X	X
SPE	serial poll enable		M	UC	Y	0	0	1	1	0	0	0	XXX	1	X	X	X	X
SRQ	service request		U	ST	X	X	X	X	X	X	X	X	XXX	X	X	1	X	X
STB	status byte	(Notes 8, 9)	M	ST	S	X	S	S	S	S	S	S	XXX	0	X	X	X	X
					8		6	5	4	3	2	1						
TCT	take control		M	AC	Y	0	0	0	1	0	0	1	XXX	1	X	X	X	X
TAG	talk address group		M	AD	Y	1	0	X	X	X	X	X	XXX	1	X	X	X	X
UCG	universal command group		M	UC	Y	0	0	1	X	X	X	X	XXX	1	X	X	X	X
UNL	unlisten		M	AD	Y	0	1	1	1	1	1	1	XXX	1	X	X	X	X
UNT	untalk	(Note 11)	M	AD	Y	1	0	1	1	1	1	1	XXX	1	X	X	X	X

The 1/0 coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

1. D1–D8 specify the device dependent data bits.
2. E1–E8 specify the device dependent code used to indicate the EOS message.
3. L1–L5 specify the device dependent bits of the device's listen address.
4. T1–T5 specify the device dependent bits of the device's talk address.
5. S1–S5 specify the device dependent bits of the device's secondary address.
6. S specifies the sense of the PPR.

S	Response
0	0
1	1

P1–P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
0	0	0	PPR1
.	.	.	.
.	.	.	.
.	.	.	.
1	1	1	PPR8

7. D1–D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
8. S1–S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)
9. The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
10. The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
11. This code is provided for system use, see 6.3.

Modem Products

4

89024
ARCHITECTURAL OVERVIEW 2400 BPS INTELLIGENT MODEM CHIP SET

- 300 to 2400 bps Full-Duplex Modem
- Operates with Public and Private Unconditioned Lines
- CCITT V.22 bis, V.22 A & B, V.21, Bell 212A, 103 Compatible
- DSP Implementation
- DTMF or Pulse Dialing with Automatic Adaptation to Network
- Call Progress Tone Detection for Most North American and European Networks
- Analog and Digital Loopback Diagnostics with Mark/Space Pattern Generation and Error Detection per V.54
- Programmable Output Level from −1 dBm to −16 dBm
- Automatic Dial and Redial Capability
- Two Chip Solution, no External Microcontroller Required
- Serial Command Set Compatible with Hayes Smartcom II Communication Software
- Easily Customized Command Set
- On-Chip 4 Wire to 2 Wire Hybrid Function with Disable Option
- On-Chip Serial Port and Handshake Signals for DTE Interface
- A Full Set of Control Signals for Telephone Line Control
- Telephone Line Audio Monitor Output
- Billing Delay Timer
- Auxiliary Relay Control Output
- Automatic Modem Type Recognition
- Low Power CHMOS/HMOS Devices

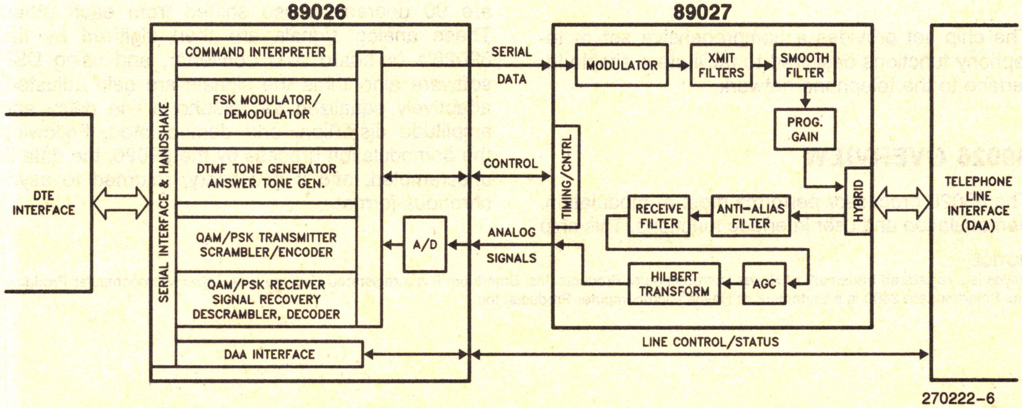


Figure 1. System Block Diagram

October 1986
Order Number: 270222-001

GENERAL DESCRIPTION

The Intel 89024 chip set is a high performance and highly integrated modem, providing a complete system in two chips. The system implements CCITT V.22 bis 2400 bps, V.22 A & B/ Bell 212A 1200 bps, and V.21/Bell 103 300 bps full-duplex modem functions. In addition to supporting a strict implementation of CCITT and Bell standards, a complete set of Hayes Smartcom II modem commands is also provided for modem configuration and user interface.

In stand-alone modem applications, the 89024 chip set along with a Data Access Arrangement (DAA) and RS-232 driver/receivers, represent the circuitry required for implementing an auto-dial, auto-answer, sync/ async, 300 to 2400 bps, full duplex modem.

In applications where user proprietary modem control commands may be required, the user can replace the 89024 internal command module with custom proprietary software resident in the 89026 microcontroller's on-chip ROM or an external memory device.

The 89024 system consists of a 16 bit application specific processor (89026) and an analog front end device (89027). The 89026 processor executes all "Digital Signal Processing" algorithms for the modem signals, as well as providing all modem control functions typically performed by an external processor. The analog front end provides the telephone line 2 wire to 4 wire interface, D/A conversion, and most of the complex filtering functions required in QAM/PSK/FSK modems. Refer to Figure 1 for a simplified block diagram of the system.

The chip set provides a comprehensive set of telephony functions designed to facilitate a simple interface to the telephone network.

89026 OVERVIEW

The 89026 processor performs most of modulation, demodulation and user interface functions. This chip is available in a standard 48 pin package. An optional 68 pin version supports an external ROM for user designed software. With this option, the signal processing algorithms resident in the 89026, can be controlled by the customer designed external software for proprietary modem control and call progress management applications. A block diagram of 89026 is provided in Figure 2.

This device contains a TTL compatible serial link to DTE/DCE equipment, along with a full complement of V.24/RS-232-C control signals. Alternatively, a UART or USART may be used to directly transfer data to/from a microcomputer bus. The device supports a complete set of Hayes compatible modem control commands. This compatibility facilitates communications between the 89024 and most PC software written for the Hayes Smartmodem 2400 product.

In the transmit direction, the 89026 synthesizes DTMF tones and the 300 bps FSK modem signal prior to transmitting them to the 89027 as digitized amplitude samples. During 1200 and 2400 bps operations, quadrature amplitude modulation (QAM) is used to send 2 or 4 bits of information at 600 baud to the 89027. Since the QAM coding technique is inherently a synchronous transmission mechanism, during a synchronous QAM transmission, the asynchronous data is synchronized by adding or deleting stop bits. Following the synchronization process, the 89026 transmits digitized phase and amplitude samples to the 89027 over a high speed serial link.

In the receive direction, the information is received by the 89026 from the 89027 as two signals which are 90 degrees phase shifted from each other. These analog signals are then digitized by the 89026's on-board A/D converter, and using DSP software algorithms the signals are gain adjusted, adaptively equalized for telephone line delay and amplitude distortion, and demodulated. Following the demodulation process by the 89026, the data is unscrambled, and if necessary, returned to asynchronous format.

NOTICE:
Hayes is a registered trademark of Hayes Microcomputer Products, Inc. Smartcom II is a registered trademark of Hayes Microcomputer Products, Inc. Smartmodem 2400 is a trademark of Hayes Microcomputer Products, Inc.

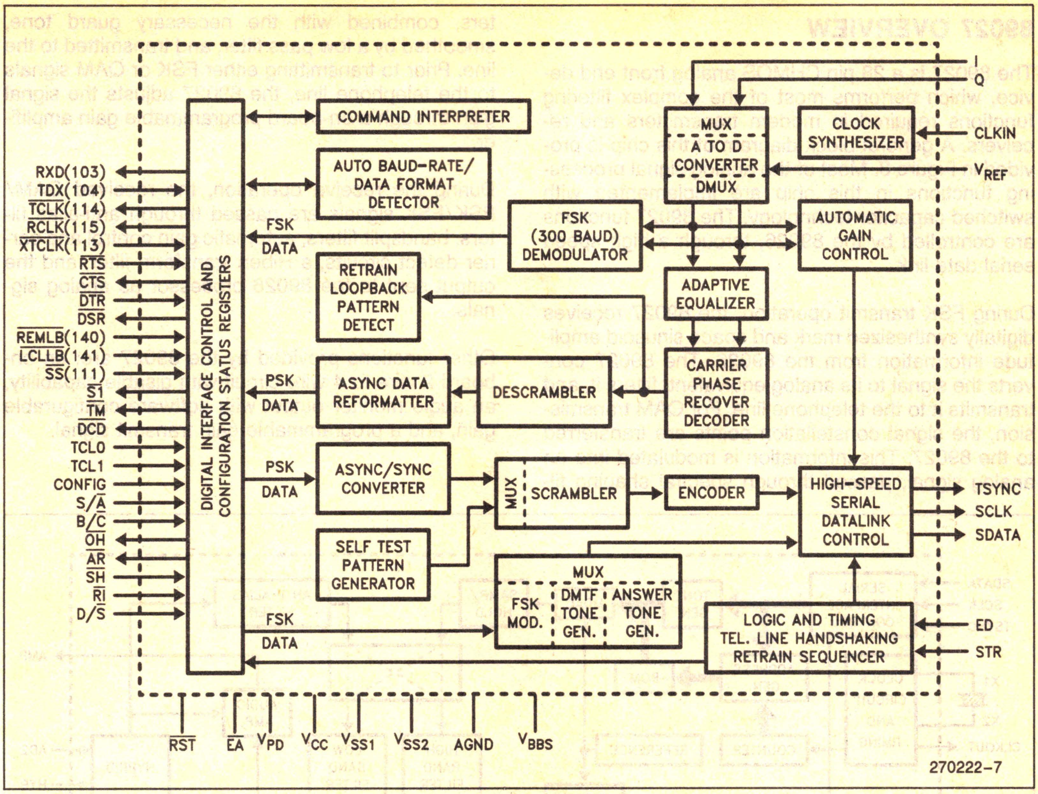


Figure 2. 89026 Block Diagram

89027 OVERVIEW

The 89027 is a 28 pin CHMOS analog front end device, which performs most of the complex filtering functions required in modem transmitters and receivers. A general block diagram of this chip is provided in Figure 3. Most of the analog signal processing functions in this chip are implemented with switched capacitor technology. The 89027 functions are controlled by the 89026, through a high speed serial data link.

During FSK transmit operation, the 89027 receives digitally synthesized mark and space sinusoid amplitude information from the 89026. The 89027 converts the signal to its analog equivalent, filters it, and transmits it to the telephone line. For QAM transmission, the signal constellation points are transferred to the 89027. This information is modulated into an analog signal, passed through spectral shaping filters, combined with the necessary guard tone, smoothed by a low pass filter, and transmitted to the line. Prior to transmitting either FSK or QAM signals to the telephone line, the 89027 adjusts the signal gain through a on-board programmable gain amplifier.

During the receive operation, the received QAM/PSK/FSK signals are passed through anti-alias filters, bandsplit filters, automatic gain control and carrier detect circuits, a Hibert transform filter, and the output sent to the 89026 processor as analog signals.

Other functions provided by the 89027 are: an on-board 2 wire to 4 wire circuit with disable capability, an audio monitor output with software configurable gain, and a programmable gain transmit signal.

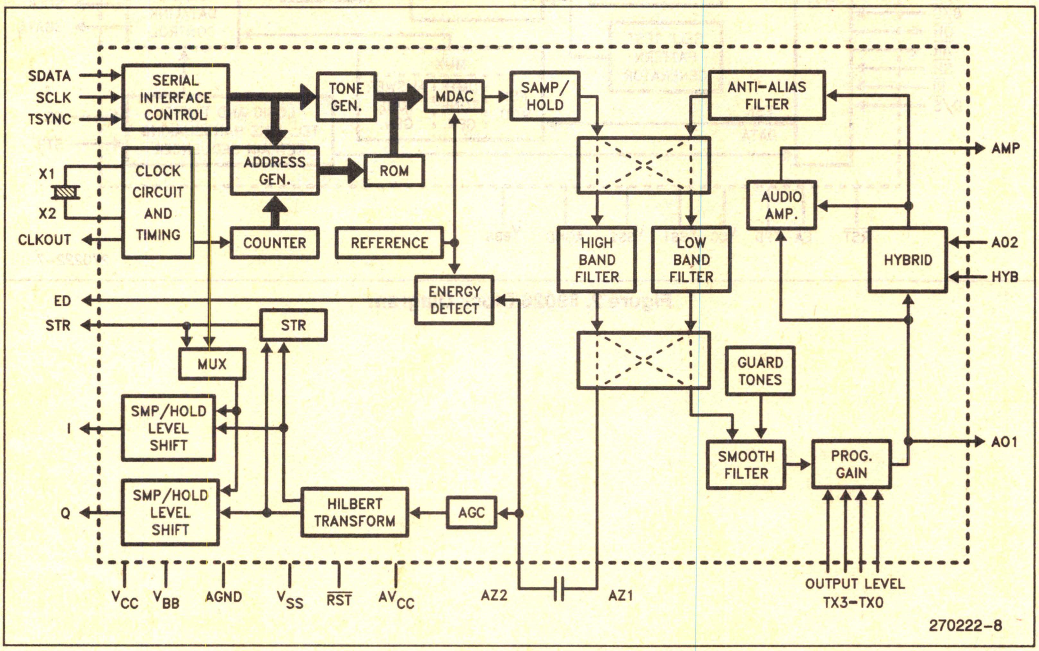


Figure 3. 89027 Block Diagram

APPLICATIONS OVERVIEW

The block diagram of a stand-alone 300 to 2400 bps Hayes compatible modem is depicted in Figure 4. The DAA section shown in this diagram may be obtained as an "FCC Part 68" approved module, or implemented using the suggested diagram in Figure 5.

In the above example, the modem conforms to CCITT and Bell data call set-up protocols, for identifying and connecting to remote modems. Because these protocols are quite different from each other and do not provide recognition of the remote modem type (i.e. V.22 bis/V.22 or 212A), the Intel chip set provides the additional capability to identify and adapt to remote modems without user intervention.

This feature is beneficial during the migration phase of the technology from 1200 bps to 2400 bps. In North America, where the installed base of 1200 bps modems is mostly made-up of 212A type, this feature allows a "Data Base Service Provider"to easily upgrade its existing 212A modems to 2400 bps V.22 bis standard, using the Intel 89024 system, entirely transparent to the current 212A users. Similarly, a user with a 89024 based modem system can automatically call data bases with either 212A or V.22 bis modems, without concern over the difference. This feature's benefits are realized in smooth upgrading of data links, with minimum cost and reduced disruption in services.

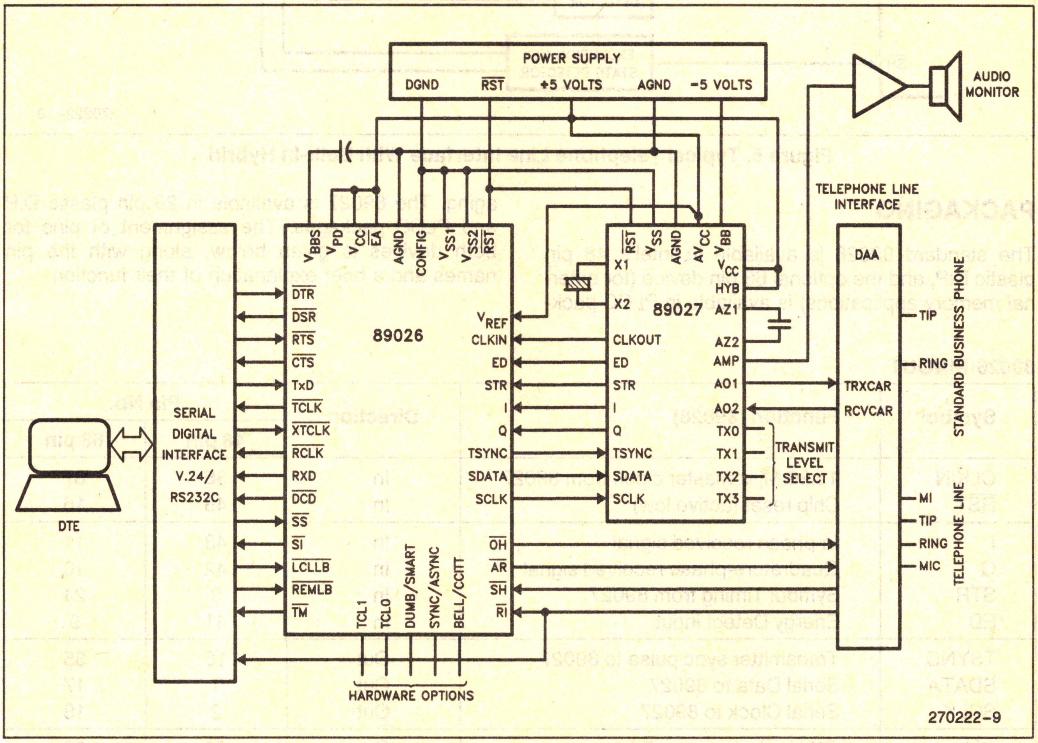


Figure 4. Typical Modem Configuration

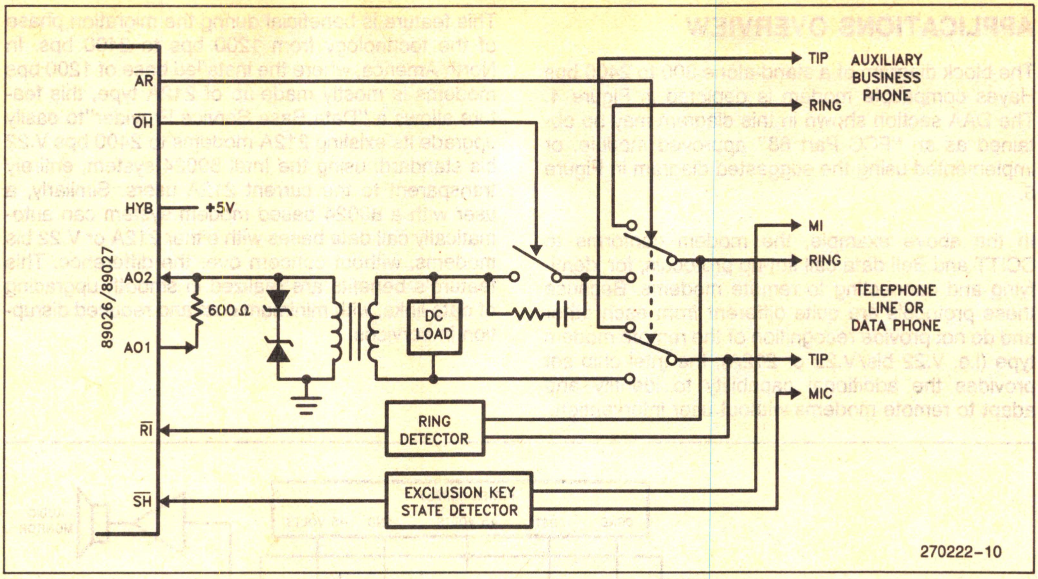


Figure 5. Typical Telephone Line Interface With Built-In Hybrid

PACKAGING

The standard 89026 is available in Intel's 48 pin plastic DIP, and the optional 68 pin device (for external memory applications) is available in PLCC packaging. The 89027 is available in 28 pin plastic DIP and PLCC packages. The assignment of pins for both devices is given below, along with the pin names and a brief explanation of their function.

89026 PINOUT

Symbol	Function (89026)	Direction	Pin No. 48 pin	Pin No. 68 pin
CLKIN	12.96 MHz master clock from 89027	In	36	67
$\overline{RST}$	Chip reset (active low)	In	48	16
I	In-phase received signal	In	43	11
Q	Quadrature-phase received signal	In	42	10
STR	Symbol Timing from 89027	In	3	24
ED	Energy Detect input	In	41	9
TSYNC	Transmitter sync pulse to 89027	Out	10	35
SDATA	Serial Data to 89027	Out	1	17
SCLK	Serial Clock to 89027	Out	2	18
$\overline{OH}$	Off-Hook control to DAA	Out	26	33
$\overline{SH}$	Switch-Hook from dataphone	In	28	44
$\overline{RI}$	Ring Indicator from DAA	In	27	42
$\overline{AR}$	Aux Relay control to DAA	Out	25	38

89026 PINOUT (Continued)

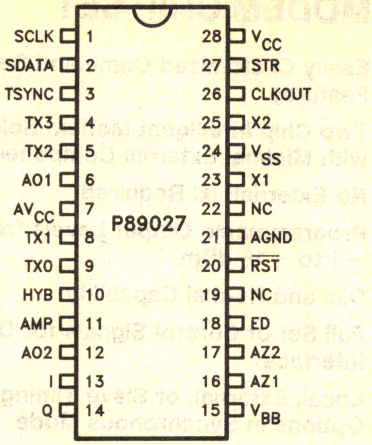
Symbol	Function (89026)	Direction	Pin No.	
			48 pin	68 pin
TCL1	Synchronous clock source select	In	23	20
TCL0	Synchronous clock source select	In	24	19
B/$\overline{C}$	BELL/$\overline{CCITT}$ default option	In	22	21
S/$\overline{A}$	Sync/$\overline{Async}$ format select	In	47	15
D/$\overline{S}$	Dumb/$\overline{Smart}$ mode select	In	32	6
CONFIG	Pin configuration sel ($\overline{48}$/68)	In	40	8
$\overline{TM}$	Test Mode Indicator	Out	13	39
TXD	Transmitted data from DTE	In	6	27
RXD	Received data to DTE	Out	8	29
$\overline{RTS}$	Request to send from DTE	In	21	22
$\overline{CTS}$	Clear to Send to DTE	Out	20	23
$\overline{DSR}$	Data Set Ready to DTE	Out	19	30
$\overline{DCD}$	Data Carrier Detect to DTE	Out	18	31
$\overline{DTR}$	Data Terminal Ready from DTE	In	4	25
$\overline{RCLK}$	Received clock to DTE	Out	9	34
$\overline{TCLK}$	Transmit clock to DTE	Out	7	28
$\overline{XTCLK}$	External timing clock from DTE	In	5	26
$\overline{SI}$	Speed Indicator to DTE	Out	17	32
$\overline{SS}$	Speed select from DTE	In	31	5
$\overline{REMLB}$	Remote Loopback Command from DTE	In	30	7
$\overline{LCLLB}$	Local Loopback Command from DTE	In	29	4
V_{CC}	Positive power supply (+5V)	+5V	38	1
V_{PD}	Ram back-up power	+5V	46	14
V_{REF}	A/D converter reference	+5V	45	13
V_{SS1}	Digital ground	GND	11	36
V_{SS2}	Digital ground	GND	37	68
AGND	Analog ground	AGND	44	12
V_{BBS}	Back-bias generator output	Out	12	37
$\overline{EA}$	Memory enable	In	39	2
AD0-AD15	External memory access address/data	I/O	–	60–45
NMI	Non-maskable Interrupt (V_{SS})(1)	In	–	3
X2	Crystal output (NC)(2)	Out	35	66
CLKOUT	Clock output (NC)(2)	Out	–	65
$\overline{TEST}$	Factory test (V_{CC})(3)	In	–	64
INST	External memory instruction fetch	Out	–	63
ALE	Address latch enable	Out	34	62
$\overline{RD}$	External memory read	Out	33	61
READY	External memory ready (V_{CC})(3)	In	16	43
$\overline{BHE}$	External memory bus high enable	Out	15	41
$\overline{WR}$	External memory write	Out	14	40

NOTES:
1. Pins marked with (V_{SS}) must be connected to V_{SS}.
2. Pins marked with (NC) are to be left unconnected.
3. Pins marked with (V_{CC}) must be connected to V_{CC}.

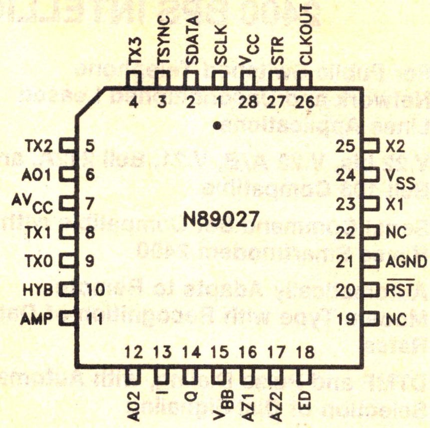
89027 PINOUT

Symbol	Function (89027)	Direction	Pin No.
V_{CC}	Positive Power Supply (Digital)	+5V	28
V_{BB}	Negative Power Supply	−5V	15
V_{SS}	Digital Ground	DGND	24
AGND	Analog Ground	AGND	21
AV_{CC}	Positive Power Supply (Analog)	+5	7
X1	Xtal Oscillator	In	23
X2	Xtal Oscillator	Out	25
CLKOUT	12.96 MHz Clock output to 89026	Out	26
$\overline{RST}$	Chip reset (active low)	In	20
HYB	Enable on-chip hybrid	In	10
AZ1	Auto-zero capacitor input	Out	16
AZ2	Auto-zero capacitor input	In	17
SDATA	Serial data from 89026	In	2
SCLK	Serial clock from 89026	In	1
TSYNC	Transmitter sync from 89026	In	3
STR	Symbol timing to 89026	Out	27
ED	Receiver energy detect to 89026	Out	18
I	In phase received signal to 89026	Out	13
Q	Quadrature-phase received signal to 89026	Out	14
AO1	Transmitter output	Out	6
AO2	Receiver input	In	12
AMP	Output to monitor speaker	Out	11
TX0	Transmitter level control	In	9
TX1	Transmitter level control	In	8
TX2	Transmitter level control	In	5
TX3	Transmitter level control	In	4

Unused pins:19, 22 must be left unconnected.

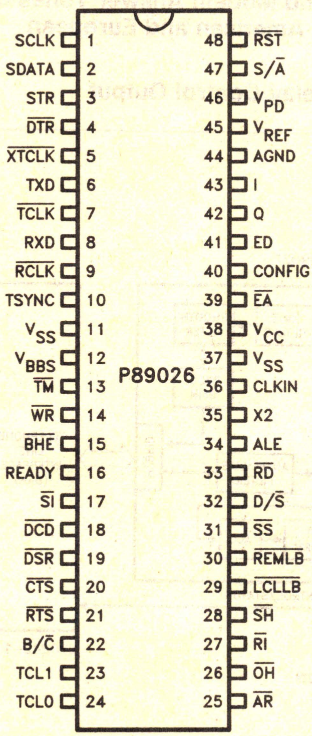


28 Pin Plastic Dip

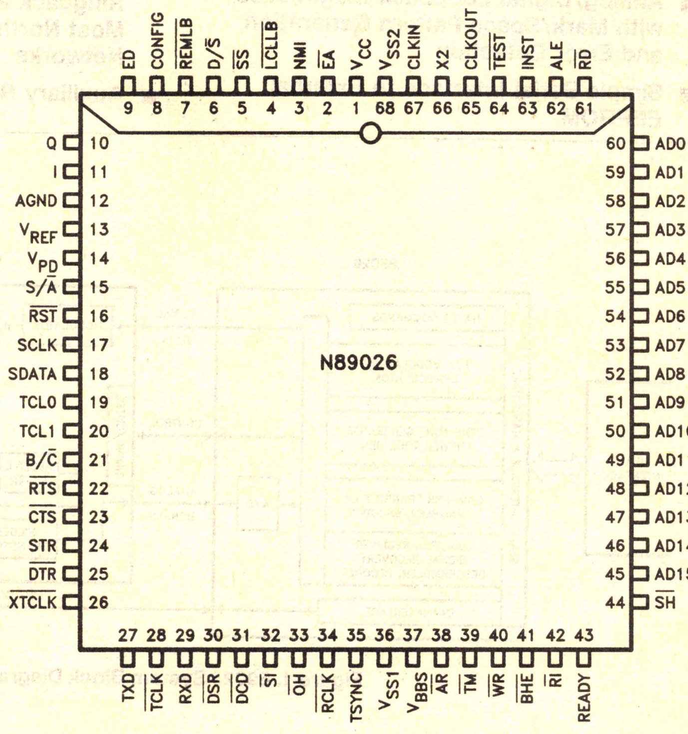


28 Pin PLCC

Figure 6a. 89027 Packages



48 Pin Plastic Dip



68 Pin PLCC

Figure 6b. 89026 Packages

89024
2400 BPS INTELLIGENT MODEM CHIP SET

- For Public Switched Telephone Network and Unconditioned Leased Lines Applications
- V.22 bis, V.22 A/B, V.21, Bell 212A, and Bell 103 Compatible
- Serial Command Set Compatible with Hayes Smartmodem 2400
- Automatically Adapts to Remote Modem Type with Recognition of Data Rates
- DTMF and Pulse Dialing, with Automatic Selection of Dial Signaling
- On-Chip Hybrid and Billing Delay Timer
- On-Chip Serial Port and Handshake Signals for RS-232/V.24 Interface
- Telephone Line Audio Monitor Output
- Analog/Digital Loopback Diagnostics with Mark/Space Pattern Generation and Error Detection
- Simple Serial Interface to External EEPROM
- Easily Customized Command Set and Features
- Two Chip Intelligent Modem Solution with Minimal External Components
- No External μC Required
- Programmable Output Levels from -1 to -16 dBm
- Dial and Re-dial Capability
- Full Set of Control Signals for DAA Interface
- Local, External, or Slave Timing Options in Synchronous Mode
- Asynchronous Character Lengths of 8, 9, 10, 11 Bits
- Adaptive Equalization
- Capable of Detecting Dial, Busy, Ringback and Modem Answer Tones of Most North American and European Networks
- Auxiliary Relay Control Output

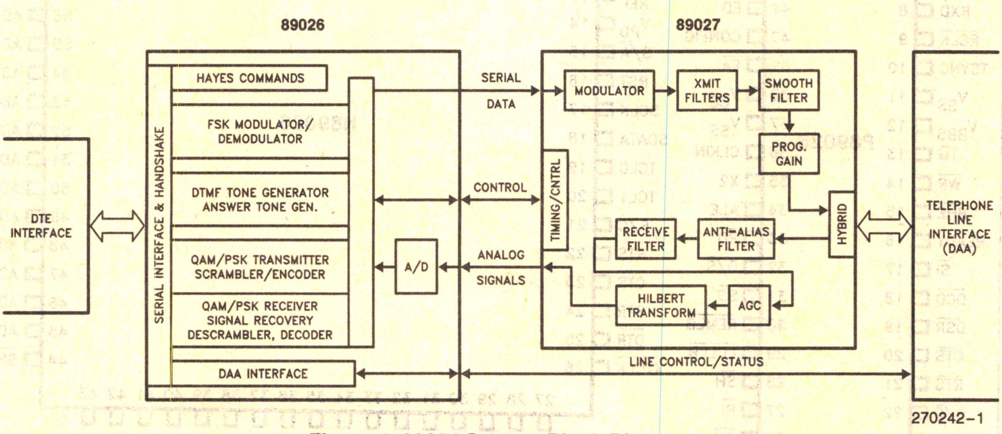


Figure 1. 89024 System Block Diagram

November 1986
Order Number: 270242-001

GENERAL DESCRIPTION

The Intel 89024 chip set is a highly integrated, high performance, intelligent modem, providing a complete system in two chips. The system conforms to the following CCITT and Bell standards:

- CCITT V.22 bis
  2400 bps sync and async
  1200 bps sync and async (fall-back)
- CCITT V.22 A & B
  1200 bps sync and async
  600 bps sync and async (fall-back)
- CCITT V.21
  0 to 300 bps anisochronous
- BELL 212A
  1200 bps sync and async
  300 bps fall-back mode
- BELL 103
  0 to 300 bps anisochronous

The 89024 system consists of a 16 bit application specific processor (89026) and an analog front end device (89027). The 89026 processor performs all "Digital Signal Processing" algorithm execution for processing the modem signals, as well as providing all modem control functions typically performed by an external processor. The analog front end provides for 2 wire and 4 wire telephone line interface, D/A conversion, and most of the complex filtering functions required in QAM/PSK/FSK modems. Refer to Figure 1 for a simplified block diagram of the system.

In stand-alone modem applications, the 89024 chip set along with a Data Access Arrangement (DAA), a serial EEPROM, and RS-232 driver/receivers, represent the circuitry required for implementing an auto-dial, auto-answer, sync/async, 300 to 2400 bps, full duplex Hayes compatible intelligent modem.

A complete set of Hayes Smartmodem 2400 commands is provided for modem configuration and user interface. Virtually all PC software written for the Hayes Smartmodem 2400 can also be used with this chip set. Alternatively, in applications where user proprietary modem control commands and features are desired, the user can replace the 89024 internal command module with custom proprietary software resident in the 89026 microcontroller's on-chip ROM or an external memory device.

The 89024 has a set of default features. Upon power up, the modem configuration will be in accordance with these default options, unless a different configuration has been saved in the external EEPROM with the &W command, or through dip switch settings. There are hardware options to select the major operating modes. These hardware straps would not normally have to be modified after initial installation, but they can be overridden with software commands.

The 89024 modem has built in auto-dialing and auto-answering capabilities. It can automatically adapt to the proper line signaling mode (Tone or Pulse), and to to the type (CCITT or Bell) and speed of the calling or answering modem. It can also detect and identify call set-up signals of telephone networks, allowing unattended data call operation.

A full set of CCITT V.54 diagnostic loop-test features is supported. The chip set also provides a line signal for audio monitoring of call progress, a comprehensive set of DAA control lines for a simple interface to the telephone network, and a full complement of TTL level RS-232/ V.24 handshake signals.

NOTICE:
Hayes is a registered trademark of Hayes Microcomputer Products, Inc.
Smartmodem 2400 is a trademark of Hayes Microcomputer Products, Inc.
Smartcom II is a registered trademark of Hayes Microcomputer Products, Inc.

PACKAGING

Both devices are available in standard DIP packages as well as PLCC packages for surface mount applications.

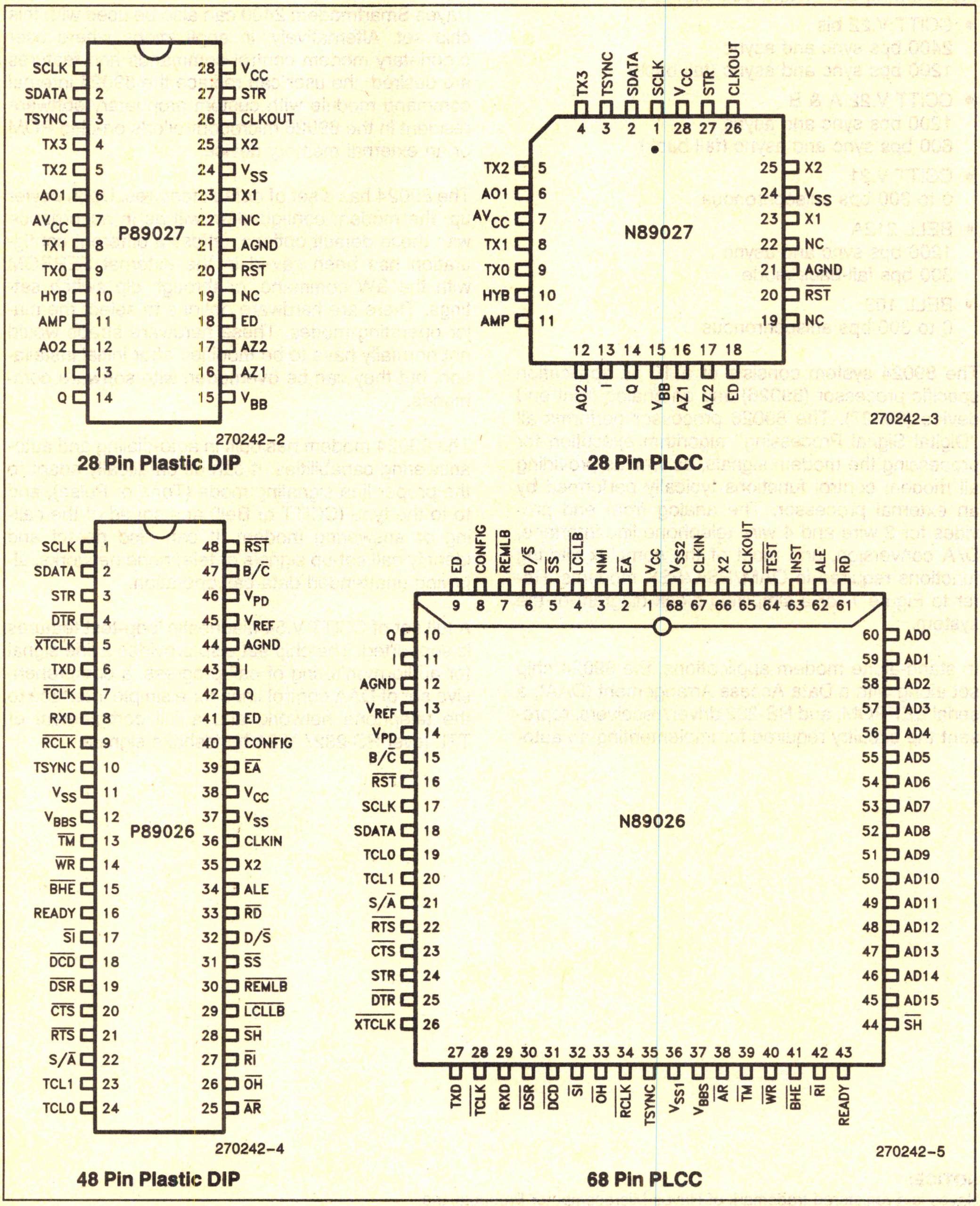


Figure 2. Device Packages

CALL ESTABLISHMENT, TERMINATION AND RETRAIN

The 89024 modem system incorporates all protocols and functions required for automatic (or manual) establishment, progress and termination of a data call.

The modem chip-set has a built in auto-dialer, both DTMF and Pulse type, and is capable of automatically adapting to the telephone dial type. The dialing sequence on the telephone link conforms to the CCITT V.25 recommendations. An exception to the V.25 is that the interrupted calling tone will not be transmitted by the calling modem, as is suggested in V.22 bis.

The modem can detect the dial, busy and ringback signals at remote end, and will provide call progress messages to the user. The modem is capable of redialing the last number dialed, by one command.

The modem when configured for auto-answer, will answer an incoming call, remain silent for the two second billing delay interval, before transmitting the answer tones. Afterwards modem to modem identification and handshaking will proceed at a speed and operating mode acceptable to both ends of the link.

The data call can also be setup by manual dialing with the modems set to data mode, or by voice to data transfer by means of mechanical switch (exclusion key), using the $\overline{SH}$ pin. Once set to data mode, the modem handshaking will proceed before the modems will be ready to accept and exchange data.

During data transmission, if one of the modems finds that the received data is likely to have a high bit error rate (indicated by a large mean square error in the adaptive equalizer), it initiates a retrain sequence. This automatic retrain feature is only available at 2400 bps, and conforms to CCITT V.22 bis recommendations.

Disconnection of the data call can be initiated by the DTE at the local end, or by the remote DTE, (if the modem is configured to accept it). Whether $\overline{DTR}$ will initiate a disconnect, depends on the last &D command. Receiving a long space from a remote modem will initiate a disconnect only after a Y1 command. The optional disconnect requests originated by the remote modem, are of two types, (1) disconnect when receiving long-space, and (2) disconnect when received carrier is dropped. The modem chip-set can also be configured to transmit 'long-space' just before disconnection, in each of the aforementioned cases.

Because the CCITT and Bell modem connection protocols are quite different from each other and do not provide recognition of remote modem type (i.e. V.22 bis to 212A), the Intel chip-set provides the additional capability of identifying the remote modem type. This feature is beneficial during the migration phase of the technology from the 1200 bps to 2400 bps. In North America, where the installed base of 1200 bps modems is mostly made-up of 212A type, this feature allows a "Data Base Service Provider" to easily upgrade the existing 212A modems to 2400 bps V.22 bis standard, transparently, to 212A users. Similarly, a user with a 89024 based modem system can automatically call data bases with either 212A or V.22 bis modems, without concern over the difference. This feature's benefits are realized in smooth upgrading of data links, with minimum cost and reduced disruption in services. Refer to Table 1 for a detailed description of remote modem compatibility.

SOFTWARE CONFIGURATION COMMANDS

This section lists the 89024 commands and registers that may be used while configuring the modem. Commands instruct the modem to perform an action, the value in the associated registers determine how the commands are performed, and the result codes returned by the modem tell the user about the execution of the commands.

The commands may be entered in a string, with or without spaces in between. Any spaces within or between commands will be ignored by the modem. During the entry of any command, the 'backspace' key (CNTRL H) can be used to correct any error. Upper case or lower case characters can be used in the commands. Commands described in the following paragraphs refer to asynchronous terminals using ASCII codes.

Table 1. Remote Modem Compatibility

Originating 89024 Modem		Answer Modem Bell 300	Bell 1200	CCITT 300	CCITT 600	CCITT 1200	CCITT 2400
Bell	300	300	300	—	—	300*	300*
	1200	1200*	1200	—	—	1200	1200
CCITT	300	—	—	300	—	—	—
	600	—	—	—	600	—	—
	1200	1200*	1200	—	—	1200	1200
	2400	1200*	1200	—	—	1200	2400

Answering 89024 Modem		Originating Modem Bell 300	Bell 1200	CCITT 300	CCITT 600	CCITT 1200	CCITT 2400
Bell	300	300	1200	—	—	1200	1200
	1200	300	1200	—	—	1200	1200
CCITT	300	—	—	300	—	—	—
	600	—	—	—	600	—	—
	1200	300*	1200	—	—	1200	1200
	2400	300*	1200	—	—	1200	2400

* These connection data rates are obtained when connecting 89024 based modems end to end. The same results may not be obtained when a 89024 based modem is connected to other modems.

Hayes Commands

AT	Attention code.
A	Go off-hook in answer mode
A/	Repeat previous command string
Bn	BELL/CCITT Protocol Compatibility at 1200 bps
Ds	The dialing commands (0-9 A B C D * # P R T S W , ; ! @)
En	Echo command (En)
Hn	Switch-Hook Control If &J1 option is selected, H1 will also switch the auxiliary relay
In	Request Product Code and Checksum
Ln	Speaker Volume
Mn	Monitor On/Off
O	On-Line
Qn	Result Codes
Sn=x	Write S Register
Sn	Read S Register
Vn	Enable Short-Form Result Codes
Xn	Enable Extended Result Code
Yn	Enable Long Space Disconnect
Z	Fetch Configuration Profile
+++	The Default Escape Code

& Commands

&C	DCD Options
&D	DTR Options
&F	Fetch Factory Configuration Profile
&G	Guard Tone
&J	Telephone Jack Selection
&L	Leased/Dial-up Line Selection
&M	Async/Sync Mode Selection
&P	Make/Break Pulse Ratio
&R	RTS/CTS Options
&S	DSR Options
&T	Test Commands
&W	Write Configuration to Non Volatile Memory
&X	Sync Clock Source
&Z	Store Telephone Number

CONFIGURATION REGISTERS

The modem stores all the configuration information in a set of registers. Some registers are dedicated to special command and function, and others are bit-mapped, with different commands sharing the register space to store the command status.

S *	Ring to Answer
S1	Ring Count. (Read Only)
S2	Escape Code Character
S3	Carriage Return Character
S4	Line Feed Character
S5	Back Space Character
S6	Wait for Dial Tone
S7	Wait for Data Carrier
S8	Pause Time for the Comma Dial Modifier
S9	Carrier Detect Response Time
S10	Lost Carrier to Hang Up Delay
S11	Not Used
S12	Escape Code Guard Time
S13	Not Used
S14 *	Bit Mapped Option Register
S15	Not Used
S16	Modem Test Options
S17	Not Used
S18 *	Test Timer
S19	Not Used
S20	Not Used
S21 *	Bit Mapped Options Register
S22 *	Bit Mapped Options Register
S23 *	Bit Mapped Options Register
S24	Not Used
S25 *	Delay to DTR (Sync Only)
S26 *	RTS to CTS Delay (Half Dup.)
S27 *	Bit Mapped Options Register

NOTE:
* These S registers can be stored in the EEPROM.

Dial Modifiers

P	Pulse Dial
R	Originate call in Answer Mode
T	Tone Dial
S	Dial a stored number
W	Wait for dial tone
,	Delay a dial sequence
;	Return to command state
!	Initiate a flash
@	Wait for quit
	If neither P or T is specified in the command string, the modem automatically selects the proper dial mode.

Example:

Terminal: AT &Z T 1 (602) 555-1212
Modem: OK

Result: Modem stores T16025551212 in the external EEPROM.

The number can be dialed from asynchronous mode by issuing the following command:

Terminal: AT DS
Modem: T16025551212

or by turning on $\overline{DTR}$ when in Synchronous Mode 2. Up to 33 symbols (dial digits and dial modifiers) may be stored. Spaces and other delimiters are ignored and do not need to be included in the count. If more than 33 symbols are supplied, the dial string will be truncated to 33.

APPLICATIONS OVERVIEW

The block diagram of a stand-alone 300 to 2400 bps Hayes compatible modem is depicted in Figure 3. The DAA section shown in this diagram may be obtained with FCC registration, or implemented using the suggested diagram in Figure 4.

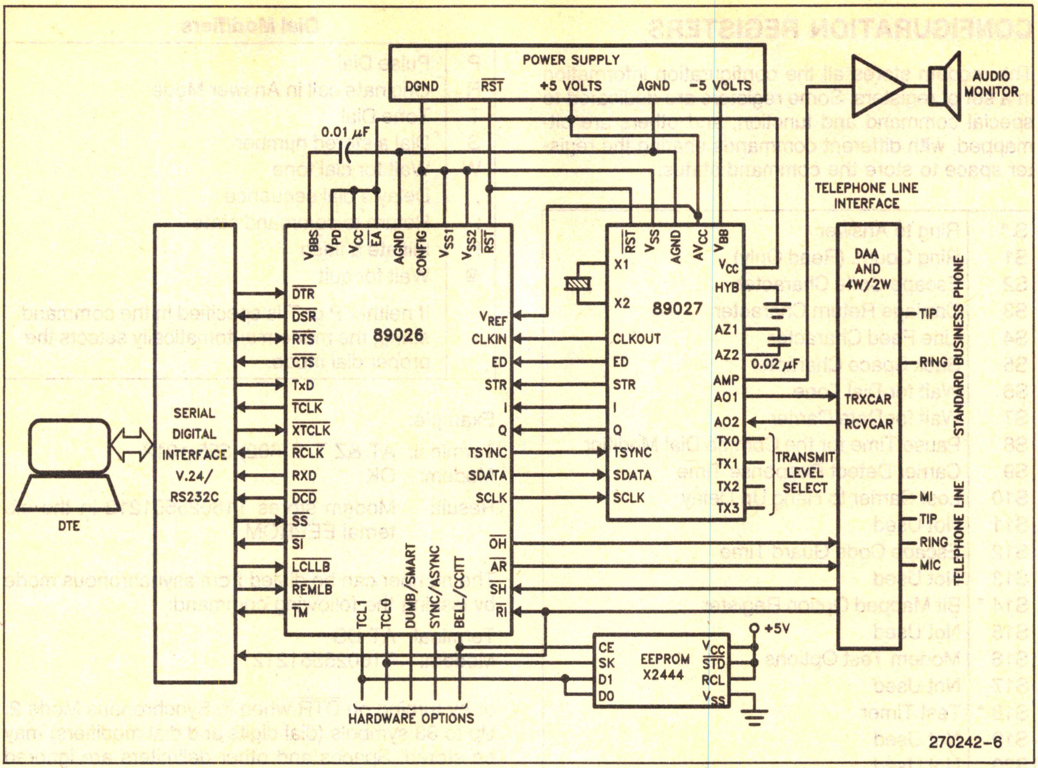


Figure 3. Typical Modem Configuration

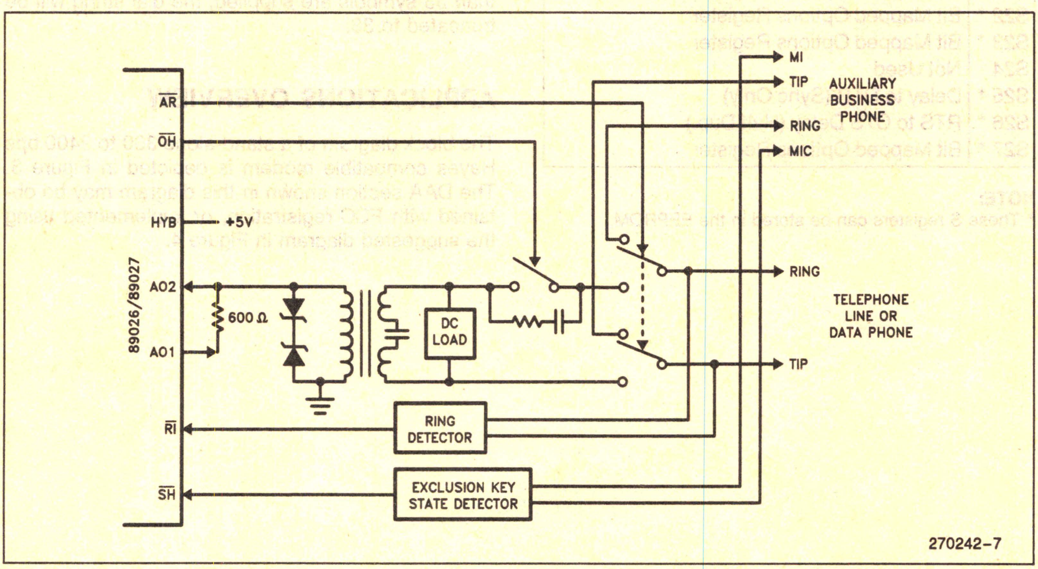


Figure 4. Typical Telephone Line Interface with Built In Hybrid

SYSTEM COMPATIBILITY SPECIFICATIONS

Parameter	Specification
Synchronous	2400 bps ±0.01% V.22 bis 1200 bps ±0.01% V.22 and 212A 600 bps ±0.01 V.22
Asynchronous	2400, 1200, 600 bps, character asynchronous. 0 - 300 bps anisochronous.
Asynchronous Speed Range	+2.3% −2.5%, extended range option of CCITT standards in character asynchronous mode.
Asynchronous Format	8,9,10,11 bits, including start, stop, parity.
Synchronous Timing Source	Internal, derived from the local oscillator. External, provided by DTE through $\overline{XTCLK}$. Slave, derived from the received clock.
Telephone Line Interface	Two wire full duplex over public switched network or 4 wire leased lines. On-chip hybrid and billing delay timers. Output level −1 to −16 dBm
Modulation	V.22 bis, 16 point QAM at 600 baud. V.22 and 212A, 4 point QAM at 600 baud. V.21 and 103, binary phase coherent FSK
Output Spectral Shaping	Square root of 75% raised cosine, QAM/PSK.
Transmit Carrier Frequencies V.22 bis, V.22, 212A V.21 at 300 bps Bell 103 mode	 Originate 1200 Hz ± .01% Answer 2400 Hz ± .01% Originate 'space' 1180 Hz ± .01% Originate 'mark' 980 Hz ± .01% Answer 'space' 1850 Hz ± .01% Answer 'mark' 1650 Hz ± .01% Originate 'space' 1070 Hz ± .01% Originate 'mark' 1270 Hz ± .01% Answer 'space' 2020 Hz ± .01% Answer 'mark' 2225 Hz ± .01%
Receive Carrier Frequencies V.22 bis, V.22, 212A V.21 Bell 103	 Originate 2400 Hz ± 7 Hz Answer 1200 Hz ± 7 Hz Originate 'space' 1850 Hz ± 12 Hz Originate 'mark' 1650 Hz ± 12 Hz Answer 'space' 1180 Hz ± 12 Hz Answer 'mark' 980 Hz ± 12 Hz Originate 'space' 2020 Hz ± 12 Hz Originate 'mark' 2225 Hz ± 12 Hz Answer 'space' 1070 Hz ± 12 Hz Answer 'mark' 1270 Hz ± 12 Hz
Receiver Sensitivity	OFF to ON threshold −45 dBm at AO2 pin ON to OFF threshold −48 dBm at AO2 pin
Line Equalization	Fixed compromise equalization, transmit. Adaptive equalizer for PSK/QAM, receive.
Diagnostics Available	Local analog loopback. Local digital loopback. Remote digital loopback. Local interface loopback modem.
Self Test Pattern Generator	Alternate 'ones' and 'zeros' and error detector, to be used along with most loopbacks. A number indicating the bit errors detected is sent to DTE.

TRANSMISSION PERFORMANCE SPECIFICATIONS

Parameter	Specification
Test condition: Unconditioned 3002 line, across the full dynamic range. The noise bandwidth is 3 KHz flat.	
Random Noise	Bit Error rate of 1 in 100000 or better at 7 dB SNR at 300 bps, 8 dB SNR at 600 bps, 10 dB SNR at 1200 bps and 17 dB SNR at 2400 bps.
Frequency Offsets(1)	± 7 Hz.
Phase Jitter(1)	2400 bps - 15° peak to peak, at up to 300 Hz. 600, 1200 bps - 45° peak to peak, at up to 300 Hz.

NOTE:
1. There is no observable data errors for the received signals, for the above limits of line impairments. These impairments are applied one at a time in absence of noise.

OTHER PERFORMANCE SPECIFICATIONS

Parameter	Min	Typ	Max	Units	Comments
DTMF Level	−16		−1	dBm	TX0−3 = 0
Tone 2nd Harmonic Distortion			−35	dB	HYB enabled into 600Ω
DTMF Twist (Balance)		3		dB	
DTMF Tone Duration	50		255	ms	Register S11
Default Duration		70		ms	
Pulse Dialing Rate		10		pps	
Pulse Dialing Make/Break		39/61 33/67		% %	US UK, Hong Kong
Pulse Interdigit Interval		785		ms	
Billing Delay Interval			2.1	sec	
Guard Tone Frequency Amplitude Frequency Amplitude		540 −3 1800 −6		Hz dB Hz dB	referenced to High channel transmit.
High Channel Transmit Amplitude		−1		dB	referenced to Low channel, Guard Tone enabled.
Guard Tone 2nd Harmonic Distortion			−50	dB	
Tone Detection Passband Frequency	290		665	Hz	3 dB Point
Tone Detection OFF to ON Threshold	−33			dBm	Into 600Ω
Tone Detection ON to OFF Threshold	−35			dBm	Into 600Ω
Dial Tone Detect Duration	3.0			sec	
Ringback Tone Detect Duration Cadence	 0.75 1.5			 sec	 Off/On Ratio
Busy Tone Detect Duration Cadence	 0.2 0.67		 1.5	 sec	 Off/On Ratio

89026 OVERVIEW

The 89026 processor performs most of modulation, demodulation and user interface functions. This chip is available in a standard 48 pin package. An optional 68 pin version supports an external ROM, for user designed software. With this option, the signal processing algorithms resident in the 89026 can be controlled by the customer designed external software for proprietary modem control and call progress management applications. A block diagram of 89026 is provided in Figure 5.

This device contains a TTL compatible serial link to DTE/DCE equipment, along with a full complement of V.24/RS-232-C control signals. Alternatively, a UART or USART may be used to directly transfer data to/from a microcomputer bus. The device supports a complete set of Hayes compatible modem control commands. This compatibility facilitates communications between the 89024 and most PC software written for the Hayes Smartmodem 2400 product.

In the transmit operation, the 89026 synthesizes DTMF tones and the 300 bps FSK modem signal prior to transmitting them to the 89027 as digitized amplitude samples. During 1200 and 2400 bps operation, quadrature amplitude modulation (QAM) is used to send 2 or 4 bits of information at 600 baud to 89027. Since the QAM coding technique is an inherently synchronous transmission mechanism, during asynchronous QAM transmission, the asynchronous data is synchronized by adding or deleting stop bits. Following the synchronization process, the 89026 transmits digitized phase and amplitude samples to 89027 over a high speed serial link.

In the receive operation, the information is received by 89026 from 89027 as two signals which are 90 degrees phase shifted from each other. These analog signals are then digitized by the 89026's onboard A/D converter, and using DSP software algorithms the signals are gain adjusted, adaptively equalized for telephone line delay and amplitude distortion, and demodulated. Following the demodulation process by the 89026, the data is unscrambled, and if necessary, returned to asynchronous format.

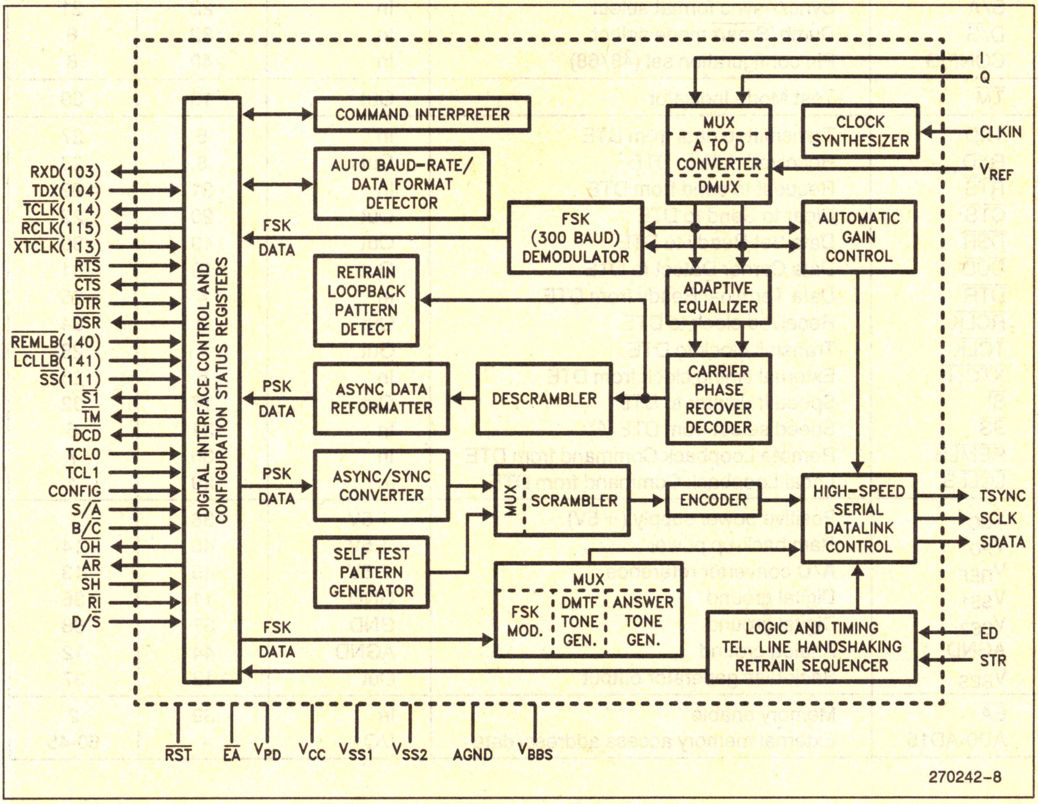


Figure 5. 89026 Block Diagram

89026 PINOUT

Symbol	Function (89026)	Direction	Pin No. 48 pin	Pin No. 68 pin
CLKIN	12.96 MHz master clock from 89027	In	36	67
$\overline{RST}$	Chip reset (active low)	In	48	16
I	In-phase received signal	In	43	11
Q	Quadrature-phase received signal	In	42	10
STR	Symbol Timing from 89027	In	3	24
ED	Energy Detect input	In	41	9
TSYNC	Transmitter sync pulse to 89027	Out	10	35
SDATA	Serial Data to 89027	Out	1	17
SCLK	Serial Clock to 89027	Out	2	18
$\overline{OH}$	Off-Hook control to DAA	Out	26	33
$\overline{SH}$	Switch-Hook from dataphone	In	28	44
$\overline{RI}$	Ring Indicator from DAA	In	27	42
$\overline{AR}$	Aux Relay control to DAA	Out	25	38
TCL1	Synchronous clock source select	In	23	20
TCL0	Synchronous clock source select	In	24	19
B/$\overline{C}$	BELL/$\overline{CCITT}$ default option	In	47	15
S/$\overline{A}$	Sync/$\overline{Async}$ format select	In	22	21
D/$\overline{S}$	Dumb/$\overline{Smart}$ mode select	In	32	6
CONFIG	Pin configuration sel ($\overline{48}$/68)	In	40	8
$\overline{TM}$	Test Mode Indicator	Out	13	39
TXD	Transmitted data from DTE	In	6	27
RXD	Received data to DTE	Out	8	29
$\overline{RTS}$	Request to send from DTE	In	21	22
$\overline{CTS}$	Clear to Send to DTE	Out	20	23
$\overline{DSR}$	Data Set Ready to DTE	Out	19	30
$\overline{DCD}$	Data Carrier Detect to DTE	Out	18	31
$\overline{DTR}$	Data Terminal Ready from DTE	In	4	25
$\overline{RCLK}$	Received clock to DTE	Out	9	34
$\overline{TCLK}$	Transmit clock to DTE	Out	7	28
$\overline{XTCLK}$	External timing clock from DTE	In	5	26
$\overline{SI}$	Speed Indicator to DTE	Out	17	32
$\overline{SS}$	Speed select from DTE (4)	In	31	5
$\overline{REMLB}$	Remote Loopback Command from DTE	In	30	7
$\overline{LCLLB}$	Local Loopback Command from DTE	In	29	4
V_{CC}	Positive power supply (+5V)	+5V	38	1
V_{PD}	Ram back-up power	+5V	46	14
V_{REF}	A/D converter reference	+5V	45	13
V_{SS1}	Digital ground	GND	11	36
V_{SS2}	Digital ground	GND	37	68
AGND	Analog ground	AGND	44	12
V_{BBS}	Back-bias generator output	Out	12	37
$\overline{EA}$	Memory enable	In	39	2
AD0-AD15	External memory access address/data	I/O	-	60-45

89026 PINOUT (Continued)

Symbol	Function (89026)	Direction	Pin No.	
			48 pin	68 pin
NMI	No-maskable Interrupt(V_{SS})(1)	In	-	3
X2	Crystal output(NC)(2)	Out	35	66
CLKOUT	Clk output (NC)(2)	Out	-	65
$\overline{\text{TEST}}$	Factory test(V_{CC})(3)	In	-	64
INST	External memory instruction fetch	Out	-	63
ALE	Address latch enable	Out	34	62
$\overline{\text{RD}}$	External memory read	Out	33	61
READY	External memory ready(V_{CC})(3)	In	16	43
$\overline{\text{BHE}}$	External memory bus high enable	Out	15	41
$\overline{\text{WR}}$	External memory write	Out	14	40

NOTES:
1. Pins marked with (V_{SS}) must be connected to V_{SS}.
2. Pins marked with (NC) are to be left unconnected.
3. Pins marked with (V_{CC}) must be connected to V_{CC}.
4. $\overline{\text{SS}}$ pin reserved for future use.

89026 PIN DESCRIPTION

$\overline{\text{XTCLK}}$

Transmitter timing from DTE, when external clock option is selected.

TXD

The serial data from DTE to be transmitted on the line. A logic 'high' is mark. In synchronous mode, 89026 samples this data on the rising edges of $\overline{\text{TCLK}}$.

$\overline{\text{TCLK}}$

Clock output from 89026 as timing source for data exchange from DTE to modem. Serial data is read on the rising edges of the $\overline{\text{TCLK}}$. This output is High in asynchronous mode.

RXD

The serial data to DTE. 'Mark' is a logic High. In synchronous mode, the rising edge of $\overline{\text{RCLK}}$ occurs in the middle of RXD.

$\overline{\text{RCLK}}$

Synchronous clock output. Rising edge of $\overline{\text{RCLK}}$ occurs in the middle of each RXD bit. This pin remains High in asynchronous mode.

V_{BBS}

This pin to be connected to AGND through a 0.01 μF capacitor.

$\overline{\text{TM}}$

A Low indicates maintenance condition in the modem.

$\overline{\text{DCD}}$

In async operation, $\overline{\text{DCD}}$ remains Low regardless of data carrier (default), or it can be programmed to indicate received carrier signal is within the required timing and amplitude limits. In sync operation Low indicates the received carrier signal is within the required timing and amplitude limits.

$\overline{\text{DSR}}$

Low indicates modem is off-hook, and it is in data transmission mode, and the answer tone is being exchanged. $\overline{\text{CTS}}$ Low indicates modem is prepared to accept data.

$\overline{\text{RTS}}$

In async mode $\overline{\text{RTS}}$ is ignored. Under command control, in sync mode $\overline{\text{RTS}}$ can be ignored, or the modem can respond with a Low on $\overline{\text{CTS}}$.

$\overline{\text{DTR}}$

&D0 command will cause the modem to ignore $\overline{\text{DTR}}$. For &D1 the modem assumes the asynchronous command state on a Low to High transition of the $\overline{\text{DTR}}$ circuit. The &D2 command does the same as &D1 except the state of $\overline{\text{DTR}}$ will enable/disable auto answer. A Low to High transition of $\overline{\text{DTR}}$ after the &D3 command will cause the modem to assume the initialization state.

B/$\overline{\text{C}}$

Low configures the modem to CCITT V.21. High will configure the modem to Bell 103, when at 300 bps speed. Also used as external EEPROM CE Signal. Refer to Figure 3.

TCL1, TCL0

TCL1	TCL0	Transmitter Timing Selection
0	0	From local crystal oscillator
0	1	From the Receive synchronous clock ($\overline{RCLK}$)
1	0	From DTE through $\overline{XTCLK}$ pin
1	1	Extended hardware option register

The extended option register mode assumes that the TCL0 and TCL1 pins are pulled high through resistors so that the 89026 can drive them Low. TCL0 is used to output a clock and serial data is read in on TCL1. Values 00, 01, and 10 on these pins are ignored when operating in asynchronous mode. These pins are also used as the serial clock and data for interface to an EEPROM. Refer to Figure 3.

$\overline{AR}$

This Auxiliary relay control is for switching a relay for voice or data calls. High is voice, Low is data.

$\overline{RI}$

A Low signal from DAA indicates line ringing. This input is ignored when the modem is configured for leased line. This signal should follow the ring cadence.

$\overline{OH}$

Low controls off hook. High indicates on hook. When dialing, this control is used to pulse dial the line.

$\overline{SH}$

Used as a telephone voice to data switch or vice versa. Any logic level transition will toggle the modem state. This input is ignored, if a software command attempts to switch the modem between voice and data.

$\overline{LCLLB}$

A Low will set the modem in the local analog loopback test mode. Logic Low levels applied simultaneously to $\overline{REMLB}$ and $\overline{LCLLB}$ pins, sets the modem to the local digital loopback.

$\overline{REMLB}$

A logic Low on this pin initiates a remote loopback condition.

$\overline{SI}$

Selects one of the two data rates or ranges of rates in the DTE to correspond to the rate in modem. High selects the higher rate (2400 CCITT/1200 BELL) or range of rates. Low selects the Low rate or range of rates.

D/$\overline{S}$

A Low on this pin will indicate the smart mode which will respond to all commands. A High will ignore all commands.

V_{REF}

Voltage reference for the analog to digital converter should be connected to the 89027 AVcc.

V_{PD}

The internal RAM power down supply voltage to be connected to 5 Volts during normal operation.

S/$\overline{A}$

A Low enables asynchronous mode, which can be overridden with the &M command. Synchronous mode 3 is enabled with a High on this pin.

CONFIG

High indicates external software is available. Low ignores external memory.

89026 ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10 to +80° C

Storage Temperature .... −40 to +125° C

Voltage from Any Pin to V_{SS} or AGND .... −0.3V to +7.0V

Average Output Current from Any Pin .... 10 mA

Power Dissipation .... 1.5 Watts

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
TA	Ambient Temperature Under Bias	0	+70	C
V_{CC}	Digital Supply Voltage	4.75	5.25	V
V_{REF}	Analog Supply Voltage	4.75 $V_{CC}-.3$	5.25 $V_{CC}+.3$	V V
FREQ	CLKIN Frequency 12.96 Mhz	−0.01%	+0.01%	
V_{PD}	Power-Down Supply Voltage	4.75	5.25	V

NOTE:
V_{BBS} should be connected to AGND through a 0.01 μF capacitor. AGND, V_{ss} and the 89027 V_{ss}, AGND must be nominally at the same potential.

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Comments
V_{IL}	Input Low Voltage	−0.3	+0.8	V	Except $\overline{RST}$
V_{IL1}	Input Low Voltage, $\overline{RST}$	−0.3	+0.7	V	
V_{IH}	Input High Voltage	2.0	$V_{cc}+.5$	V	Except $\overline{RST}$, NMI, CLKIN
V_{IH1}	Input High Voltage, $\overline{RST}$ Rising	2.4	$V_{cc}+.5$	V	
V_{IH2}	Input High Voltage, $\overline{RST}$ Falling	2.1	$V_{cc}+.5$	V	
V_{IH3}	Input High Voltage, NMI, CLKIN	2.4	$V_{cc}+.5$	V	
V_{OL}	Output Low Voltage		0.45	V	See Note 1.
V_{OH}	Output High Voltage	2.4		V	See Note 2.
I_{CC}	V_{cc} Supply Current		200	mA	All outputs disconnected
I_{PD}	V_{PD} Supply Current		1	mA	Normal operation and Power-Down
I_{REF}	V_{REF} Supply Current		15	mA	
I_{LI}	Input Leakage Current		±10	μA	$V_{in}=0$ to V_{cc} See Note 3
I_{IH}	Input High Current to $\overline{EA}$		100	μA	$V_{IH}=2.4V$
I_{IL}	Input Low Current		−100	μA	$V_{IL}=0.45V$ See Note 4
I_{IL1}	Input Low Current to $\overline{RST}$		−2	mA	$V_{IL}=0.45V$
I_{IL2}	Input Low Current S/$\overline{A}$, $\overline{SH}$, $\overline{RI}$, READY		−50	μA	$V_{IL}=0.45V$
C_s	Pin Capacitance (Any Pin to V_{ss})		10	pF	1 MHz

NOTES:
1. I_{OL} = 0.36 mA for pins TCL0, TCL1, B/$\overline{C}$, $\overline{RTS}$, $\overline{CTS}$, $\overline{DSR}$, $\overline{DCD}$, $\overline{SI}$, $\overline{AR}$, and $\overline{OH}$. Also if AD0 - AD15 are configured as I/O ports.
I_{OL} = 2.0 mA for $\overline{TM}$, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, RXD, $\overline{TCLK}$, and AD0 - AD15 when used as external memory bus.
2. I_{OH} = −20 μA for pins TCL0, TCL1, B/$\overline{C}$, $\overline{RTS}$, $\overline{CTS}$, $\overline{DSR}$, $\overline{DCD}$, $\overline{SI}$, $\overline{AR}$, and $\overline{OH}$.
I_{OH} = −200 μA for $\overline{TM}$, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, RXD, $\overline{TCLK}$, and AD0 - AD15 when used as external memory bus. AD0 - AD15 when used as I/O ports, have open-drain outputs.
3. For pins $\overline{DTR}$, $\overline{XTCLK}$, TXD, D/$\overline{S}$, $\overline{SS}$, $\overline{REMLB}$, $\overline{LCLLB}$, CONFIG, AD0-AD15.
4. TCL0, TCL1, B/$\overline{C}$, $\overline{RTS}$.
5. Power must be applied to the device in the following sequence: V_{ss} first, then V_{cc}.

AC CHARACTERISTICS (V_{CC}, V_{PD} = 4.75 to 5.25 Volts; T_A = 0°C to 70°C; CLKIN = 12.96 MHz)

Test Conditions: Load capacitance on output pins = 80 pF
Frequency = 12.96 MHz

Timing Requirements

Symbol	Parameter	Min	Max	Units
TAVDV	Address Valid to Input Data Valid		5Tosc − 90	ns
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc − 60	ns
TRXDX	Data Hold after $\overline{RD}$ Inactive (1)	0		ns
TRXDZ	$\overline{RD}$ Inactive to Input Data Float (1)		Tosc − 20	ns

Timing Responses

Symbol	Parameter	Min	Max	Units
FXTAL	CLKIN Frequency	−0.01%	+0.01%	
Tosc	CLKIN Period	77		ns
TCHCH	CLKOUT Period (1)	3Tosc (2)	3Tosc (2)	ns
TCHCL	CLKOUT High Time	Tosc − 20	Tosc + 20	ns
TCLLH	CLKOUT Low to ALE High	−5	20	ns
TLLCH	ALE Low to CLKOUT High	Tosc − 20	Tosc + 40	ns
TLHLL	ALE Pulse Width	Tosc − 25	Tosc + 15	ns
TAVLL	Address Setup to End of ALE	Tosc − 50		ns
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ active	Tosc − 20		ns
TLLAX	Address Hold after End of ALE	Tosc − 20		ns
TWLWH	$\overline{WR}$ Pulse Width	2Tosc − 35		ns
TQVWX	Output Data Setup to End of $\overline{WR}$	2Tosc − 60		ns
TWXQX	Output Data Hold after End of $\overline{WR}$	Tosc − 25		ns
TWXLH	End of $\overline{WR}$ to next ALE	2Tosc − 30		ns
TRLRH	$\overline{RD}$ Pulse Width	3Tosc − 30		ns
TRHLH	End of $\overline{RD}$ to next ALE	Tosc − 25		ns

NOTES:
(1) This specification is not tested, but is verified by design analysis and/or derived from other tested parameters.
(2) CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 10 nsec if Tosc is constant and the rise and fall times on XTAL are less than 10 nsec.

WAVEFORM

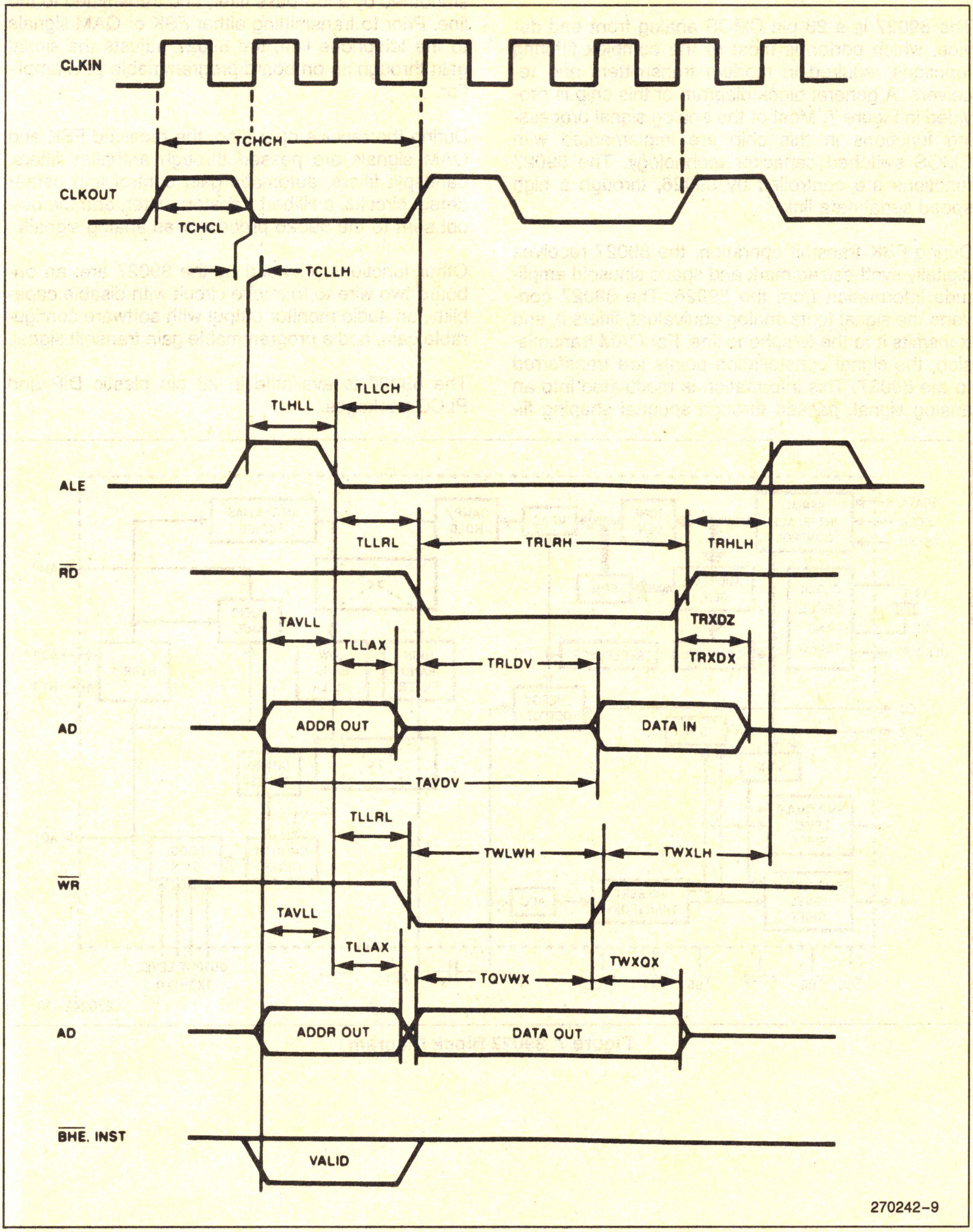


Figure 6. Bus Signal Timings

89027 OVERVIEW

The 89027 is a 28 pin CMOS analog front end device, which performs most of the complex filtering functions required in modem transmitters and receivers. A general block diagram of this chip is provided in Figure 7. Most of the analog signal processing functions in this chip are implemented with CMOS switched capacitor technology. The 89027 functions are controlled by 89026, through a high speed serial data link.

During FSK transmit operation, the 89027 receives digitally synthesized mark and space sinusoid amplitude information from the 89026. The 89027 converts the signal to its analog equivalent, filters it, and transmits it to the telephone line. For QAM transmission, the signal constellation points are transferred to the 89027. This information is modulated into an analog signal, passed through spectral shaping filters, combined with the necessary guard tone, smoothed by a low pass filter, and transmitted to the line. Prior to transmitting either FSK or QAM signals to the telephone line, the 89027 adjusts the signal gain through an on-board programmable gain amplifier.

During the receive operation, the received FSK and QAM signals are passed through anti-alias filters, bandsplit filters, automatic gain control and carrier detect circuits, a Hilbert transform filter, and the output sent to the 89026 processor as analog signals.

Other functions provided by the 89027 are: an onboard two wire to four wire circuit with disable capability, an audio monitor output with software configurable gain, and a programmable gain transmit signal.

The 89027 is available in 28 pin plastic DIP and PLCC packages.

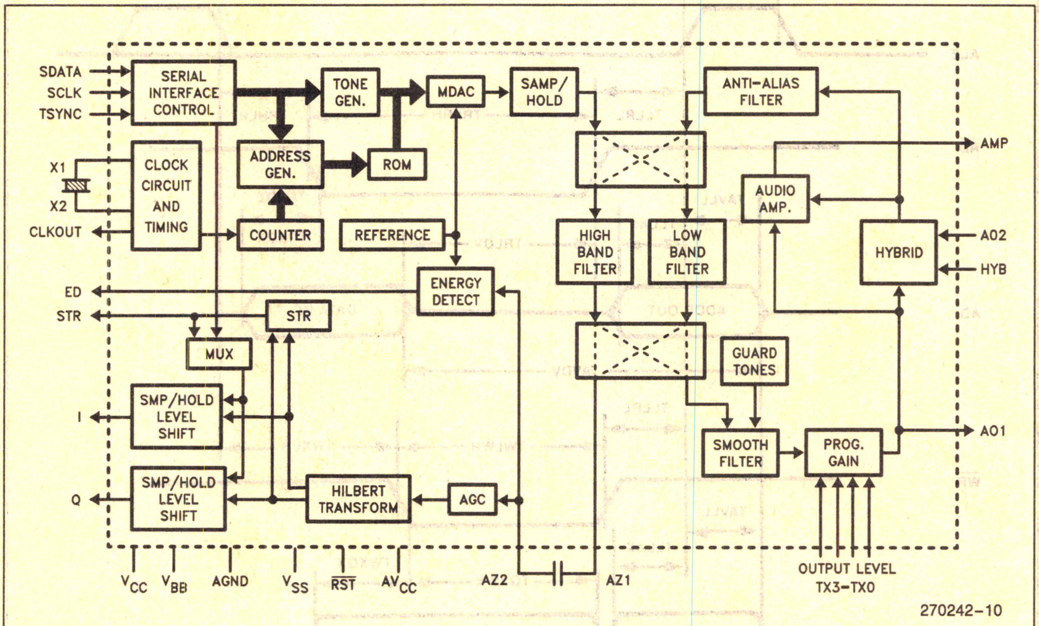


Figure 7. 89027 Block Diagram

89027 PINOUT

Symbol	Function (89027)	Direction	Pin No.
V_{CC}	Positive Power Supply (Digital)	+5V	28
V_{BB}	Negative Power Supply	−5V	15
V_{SS}	Digital Ground	DGND	24
AGND	Analog Ground	AGND	21
AV_{CC}	Positive Power Supply (Analog)	+5	7
X1	Xtal Oscillator	In	23
X2	Xtal Oscillator	Out	25
CLKOUT	12.96 MHz Clock output to 89026	Out	26
$\overline{RST}$	Chip reset (active low)	In	20
HYB	Enable on-chip hybrid	In	10
AZ1	Auto-zero capacitor	Out	16
AZ2	Auto-zero capacitor	In	17
SDATA	Serial data from 89026	In	2
SCLK	Serial clock from 89026	In	1
TSYNC	Transmitter sync from 89026	In	3
STR	Symbol timing to 89026	Out	27
ED	Receiver energy detect to 89026	Out	18
I	In phase received signal to 89026	Out	13
Q	Quadrature-phase received signal to 89026	Out	14
AO1	Transmitter output	Out	6
AO2	Receiver input	In	12
AMP	Output to monitor speaker	Out	11
TX0	Transmitter level control (LSB)	In	9
TX1	Transmitter level control	In	8
TX2	Transmitter level control	In	5
TX3	Transmitter level control (MSB)	In	4

Unused pins: 19, 22 must be left unconnected.

89027 Pinout Description

TX0-3

These four pins control the transmitted signal level. The output level can be adjusted from −1 dBm to −16 dBm in 1 dB steps.

HYB

This pin enables the on-chip hybrid.

AO1

Transmitter output.

AO2

Receiver input.

AMP

This output can be used to monitor the call progress tones and operation of the line.

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias .... −10 to +80° C

Storage Temperature .... −40 to +125° C

All Input and Output Voltages
with Respect to V_{BB} .... −0.3V to +13.0V

All Input and Output Voltages
with Respect to V_{CC} & AV_{CC}..... −13.0V to 0.3V

Power Dissipation 1.35W

Voltage with Respect
to V_{SS}(1) .... −0.3V to 6.5V

NOTE:
1. Applies to pins SCLK, SDATA, TSYNC, $\overline{RST}$, HYB, TX0–TX3 only.

POWER DISSIPATION Ambient Temp = 0° to 70° C, V_{CC} = AV_{CC} = 5 ± 5%, V_{SS} = AGND = 0.

Symbol	Parameter	Min	Typ	Max	Units
Iccl	V_{CC} Operating Current		15		mA
Ibbl	V_{BB} Operating Current		15		mA
Icco	V_{CC} Standby Current		2		mA
Oboe	V_{BB} Standby Current		2		mA
Pdo	Standby Power Dissipation		20		mW
Pdi	Operating Power Dissipation		150		mW

DC CHARACTERISTICS (Ta=0°C to 70°C, V_{CC} = +5, V_{BB} = −5, AGND = V_{SS} = 0V), supply voltage must be at the same potential as the 89026 power supply. Typical Values are for Ta = 25°C and nominal power supply values. Power must be applied in the following sequence: V_{SS}, AGND, V_{BB}, V_{CC}, and AV_{CC}. V_{CC}, AV_{CC} and 89206 V_{REF} must be nominally at the same potential.

Digital Inputs: TX0, TX1, TX2, TX3, HYB, $\overline{RST}$
Digital Outputs: CLKOUT

Symbol	Parameter	Min	Typ	Max	Units	Test Condition
Iil	Input Leakage Current	−10		+10	μA	$V_{BB} \leq Vin \leq V_{CC}$
Vil	Input Low Voltage	V_{SS}		0.8	V	
Vih	Input High Voltage	2.0		V_{CC}	V	
Vol	Output Low Voltage			0.4	V	Iol ≥ −1.6mA, 1 TTL load
Voh	Output High Voltage	2.4			V	Ich ≤ 50μa, 1 TTL load
Vcc1	CLKOUT Low Voltage			0.4	V	C1 = 60 pF
Vcoh	CLKOUT High Voltage	2.4			V	Cl = 60 pF

AC CHARACTERISTICS (Ta = 25°C, V_{CC} = AV_{CC} = 5 V, V_{SS} = AGND = 0, V_{BB} = −5 V)

ANALOG INPUTS: AO2

Parameter	Min	Typ	Max	Units	Test Condition
AO2 Input Voltage Range			−9	dBm	
AO2 Input Resistance		10		Mohms	−3.5V < Vin < +3.5V
AO2 Allowed DC offset	−30		+30	mV	Relative to AGND

AUTO ZERO CAPACITANCE

Capacitance = 0.02 μF
Tolerance = ±10 %
Voltage Rating = 10V
Type = Non-Electrolytic, low leakage.

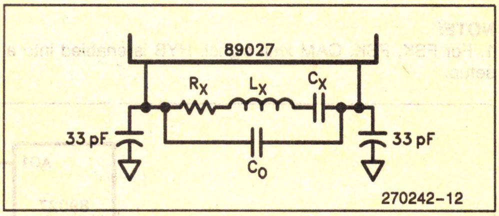


Figure 8. Crystal Equivalent Circuit

CRYSTAL REQUIREMENTS

Parameter	Min	Typ	Max	Unit	Comments
Frequency	−0.001%		+0.001%	12.960 Mhz	Refer to Figure 8
Temp. Drift (0°C–+70°C)			0.002%		
Rx		12		ohms	
Cx		0.024		pF	
Co		5.6		pF	

Crystal Type: Parallel Resonant

ANALOG OUTPUTS: A01, AMP

Parameter	Min	Typ	Max	Units	Test Condition
Load Resistance					
A01	600			ohms	
AMP	10			Kohms	
Load Capacitance					
AMP			100	pF	
Audio Amp Output Level		3		Vp	Controlled
		1.5		Vp	via the Hayes
		0.75		Vp	commands.
			30	mV	No signal input

TRANSMIT LEVEL

OUTPUT TRANSMIT LEVEL (1)	TX 3,2,1,0	Typ	Units
	0000	−1	dBm
	0001	−2	dBm
	•	•	•
	•	•	•
	•	•	•
	1110	−15	dBm
	1111	−16	dBm

NOTE:
1. For FSK, PSK, QAM xmit signal. HYB is enabled into a 600Ω network and measured at A02. Refer to Figure 9 for test setup.

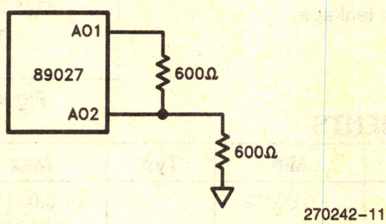


Figure 9. Transmitter Test Setup

iATC Advanced Telecommunications Components for Analog and ISDN Applications

5

SLD INTERFACE SPECIFICATION

The Subscriber Line Datalink is a three wire interface for synchronous data transfer between master and slave devices. Four full duplex time-multiplexed 64 Kbps channels are supported on a serial ping-pong link. Each channel transfers a byte of data every 125 μs.

The SLD interface was developed primarily for telecommunications applications, where 8 KHz synchronous data transfers are the norm. It provides a standardized physical interface for the transfer of circuit switched voice and data, signalling, and control channels to and from the individual subscriber circuits, physically (but not logically) isolating the interface to the per-line components from the system backplane, which is often a proprietary arrangement. This allows a large selection of different subscriber devices to be produced economically, with a standard interface, and connected to the backplane through a linecard controller, or interface controller.

INTERFACE DESCRIPTION

The three wires of the SLD interface consist of a data clock (SCL), a data direction signal (SDIR), and a ping-pong data lead (SLD). The data clock and direction signals can be common to all slave devices connected to the interface controller. A separate SLD line is connected to each slave device. The slave devices on this interface receive the clock signals from the controller, and only drive the data line when so indicated by the direction signal. The controller generates both the SDIR direction signal and the SCL data clock, which are derived from the system clock.

The SLD line itself supports a 512 Kpbs rate, as defined by the SCL clock. The data on this line is formatted as 32 bits of receive (towards slave) data and 32 bits of transmit (from slave) data. This pattern is repeated at an 8 KHz rate, with the direction of transmission being defined by the 8 KHz SDIR signal. When SDIR is high, data is transferred to the slave device, and when SDIR is low, data is transferred back to the master. Hence, the SDIR signal has a duty cycle of approximately 50%. The transmit and receive direction data is further divided into eight bytes, four transferred in each direction. The effective data rate over the SLD interface is 256 Kbps in each direction. Because all SLD lines handled by a controller share the same direction signal, these separate links are all synchronous. The exact use of the data channels on the SLD is determined by the devices connected to it.

USE OF THE SLD BY THE iATC 29CX FAMILY OF COMPONENTS

The SLD interface provides the data link to the per-line components of the 29CX family. The iATC 2952 linecard controller operates as an SLD master, and controls eight SLD interface lines. The 2952 can assign timeslots to the first two bytes of data in each direction and thus route this data to or from either of the two TDM backplane highways it supports. These bytes are typically used for subscriber voice and/or circuit switched data. The remaining two bytes in each direction can be used for commands, configuration data, signalling, status, or additional subscriber data, depending on the devices using the SLD. The 2952 routes data to/from the third and seventh bytes through its bidirectional FIFO, or via the μP port, and these channels are usually used for control. The fourth and eighth bytes are stored intermediately in holding registers by the 2952, and are typically used for signalling information.

Figure 1 shows the SLD configuration in the analog subscriber case, when the 2952 is connected to the 29C51. The first and fifth bytes (channel A) represent the primary voice channel, and route the PCM voice byte between the backplane and the 29C51 via the linecard controller. The second and sixth bytes (channel B) contain PCM data to support the secondary analog channels of the 29C51, and three party conferencing. The third byte is used as a control channel to program the features of the 29C51, while the seventh byte can be used to read back, or verify, this control information. The fourth and eighth bytes are used to transport signalling information to and from the 29C51. The SLD link provides an efficient means of routing all this information between the 29C51 combos and the linecard controller.

The SLD approach extends to support digital subscriber components as well. The first two bytes in each direction support voice/data information (B1- and B2-channels of the CCITT model). The remaining two bytes in each direction are used for control, status, and data (D-channel) as needed for the application. In a terminal application the SLD can be used to interface the digital transceiver to an SLD combo

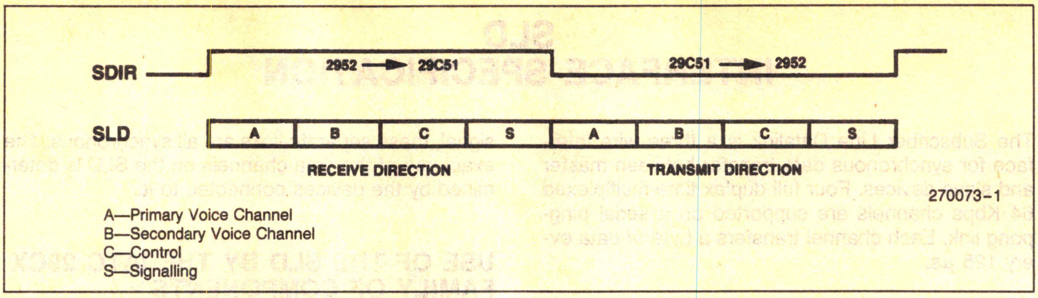


Figure 1. SLD for the Analog Subscriber Case

to provide voice capability. It can also function as a general serial data port for access to data carried in the B1-, B2- or D-channels.

SLD TIMING

The duty cycle of the 512 KHz SCL clock is typically 50%. However, because this signal is usually derived from the system clock, it may not be practical to generate a 50% duty cycle. For example, the 2952 generates a 33% duty cycle clock if the system clock is 1.536 MHz (24 timeslot systems). All slave devices built to interface to the SLD should accept duty cycles of from 30% to 70%.

A special case exists for systems with a 1.544 MHz clock. It is not possible to derive a 512 KHz SCL clock from this system clock. For this case, SCL is allowed to have an instantaneous bit rate of 514.67 KHz, with a stretched clock cycle inserted to achieve a 512 Kbps average over the 125 μs frame. This stretched clock cycle may occur as the last clock of the frame, or as the first clock of the frame (see Figure 2).

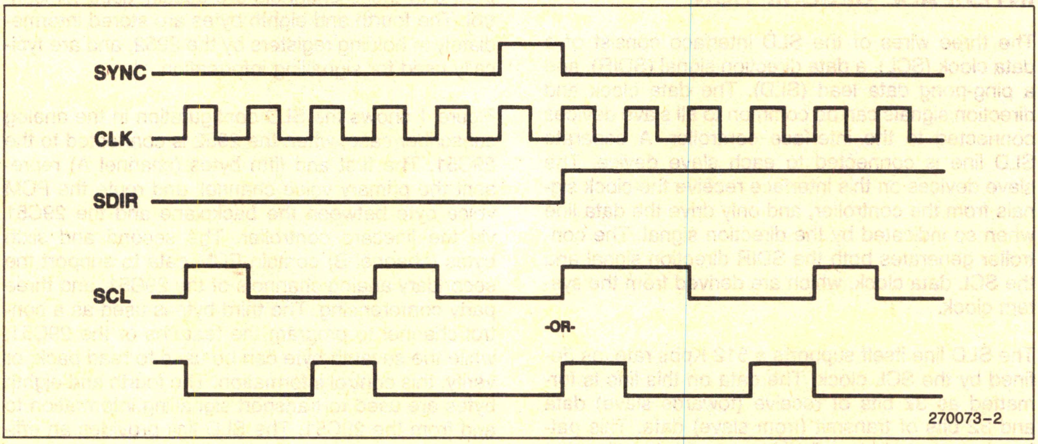


Figure 2. SLD Timing for 1.544 MHz

GENERAL SLD TIMING SPECIFICATION

Following is a general SLD timing specification which can be used as a guideline in the design of SLD master or slave devices. It allows for data reception by the master on rising or falling edges of SCL and for data reception by the slave on falling edges, and if adhered to, is compatible to all currently existing SLD standard Intel components.

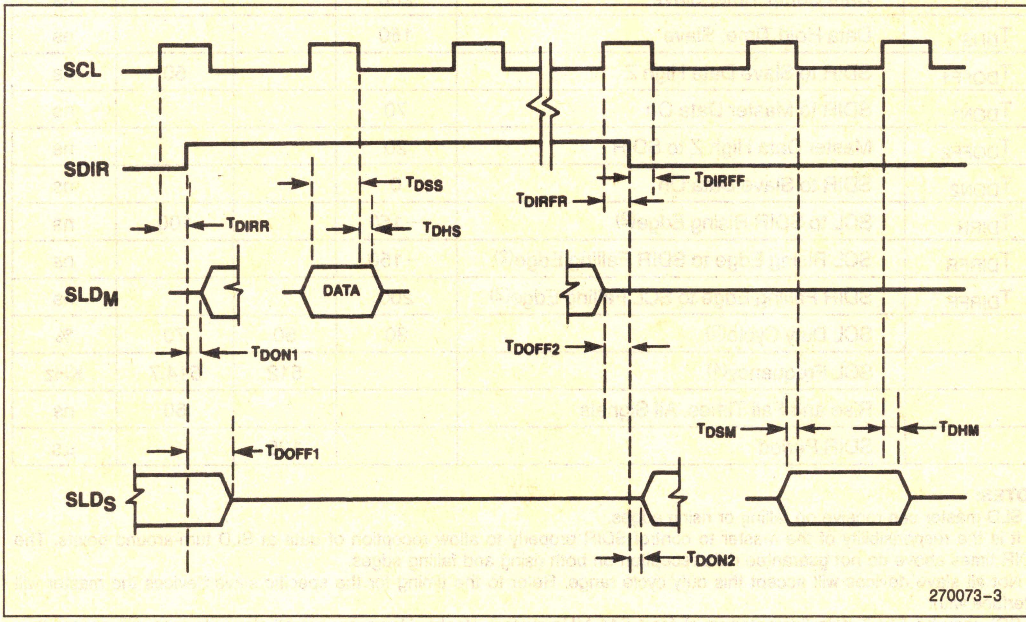


Figure 3. General SLD Timing

General SLD Timing

Symbol	Parameter	Min	Typ	Max	Unit
T_{DSM}	Data Setup Time, Master(1)	150			ns
T_{DHM}	Data Hold Time, Master(1)	0			ns
T_{DSS}	Data Setup Time, Slave	200			ns
T_{DHS}	Data Hold Time, Slave	150			ns
T_{DOFF1}	SDIR to Slave Data High Z			50	ns
T_{DON1}	SDIR to Master Data On	70			ns
T_{DOFF2}	Master Data High Z to SDIR	20			ns
T_{DON2}	SDIR to Slave Data On	0			ns
T_{DIRR}	SCL to SDIR Rising Edge(2)	−150		100	ns
T_{DIRFR}	SCL Rising Edge to SDIR Falling Edge(2)	−150			ns
T_{DIRFF}	SDIR Falling Edge to SCL Falling Edge(2)	200			ns
	SCL Duty Cycle(3)	30	50	70	%
	SCL Frequency(4)		512	514.7	KHz
	Rise and Fall Times, All Signals			50	ns
	SDIR Period		125		μs

NOTES:

1. SLD master can receive on falling or rising edges.
2. It is the responsibility of the master to control SDIR properly to allow reception of data at SLD turn-around points. The TDIR times above do not guarantee data reception on both rising and falling edges.
3. Not all slave devices will accept this duty cycle range. Refer to the timing for the specific slave devices the master will interface with.
4. SCL may be 514.7 KHz (instantaneous) for 1.544 MHz system clocks. However, not all slave devices will accept this. Refer to the timing for the specific slave devices the master will interface with. SCL must have 64 pulses per SDIR cycle in any case.

iATC 29C48
FEATURE CONTROL COMBO

- **External and User Programmable Transmit and Receive Gain**
- **Programmable External Hybrid Balance Network Select**
- **Programmable Analog, Digital, and Subscriber Loopback**
- **Programmable μ/A-Law Select**
- **Secondary Analog Input Channel**
- **Low Power Consumption**
- **External Tone Injection to Receive Path**
- **SLD A/B Channel Select (for 16 Channel Line Cards)**

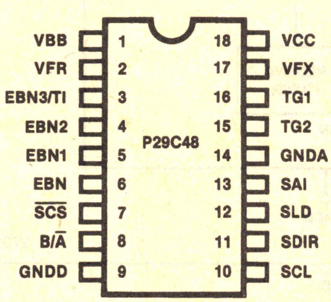
The Intel iATC 29C48 Feature Control Combo is a low cost, user-programmable, fully integrated PCM Codec with transmit/receive filters fabricated in a CMOS technology. This technology is built on CHMOS and will allow the 29C48 to realize the same excellent transmission performance as in the Intel 2913/2914 combo while achieving the low power consumption typical of CMOS circuits.

The 29C48 supports the analog subscriber with a variety of added per-line features to the normal BORSCHT functions associated with the analog line circuit. Some of these features include secondary analog input channel, programmable transmit and receive gain, custom hybrid balancing network selection, and programmable μ or A-law conversions. Additionally, the 29C48 can operate on either the A or B channel of the SLD interface, allowing two combos to be connected to one SLD link. In order to facilitate the SLIC interface in this configuration, the 29C48 generates chip select signals for the proper routing of signaling information.

A unique feature of the 29C48 is programmable tone injection. This feature and its SLD interface makes it particularly easy to use in conjunction with Intel's advanced tranceivers, such as the iATC 29C53, in subscriber equipment environments.

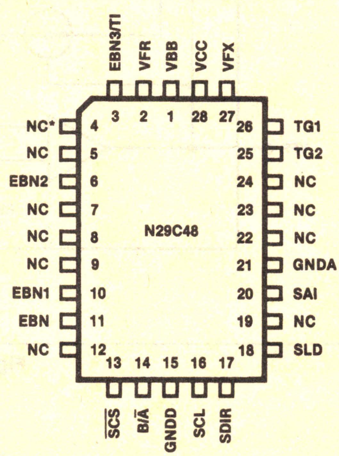
The 29C48 is intended for use with the 2952 Integrated Line Card Controller in digital switching environments. These components allow the system transmit and receive backplane highways to operate at different frequencies from that of the subscriber interface data channels. The 2952 handles the transfer of voice and feature control information between the backplane and the 29C48.

18-Lead Plastic Dual-In-Line Package



270153–1

Plastic Leaded Chip Carrier



*Not connected

Figure 1. Pin Configurations

270153–26

November 1986
Order Number: 270153-002

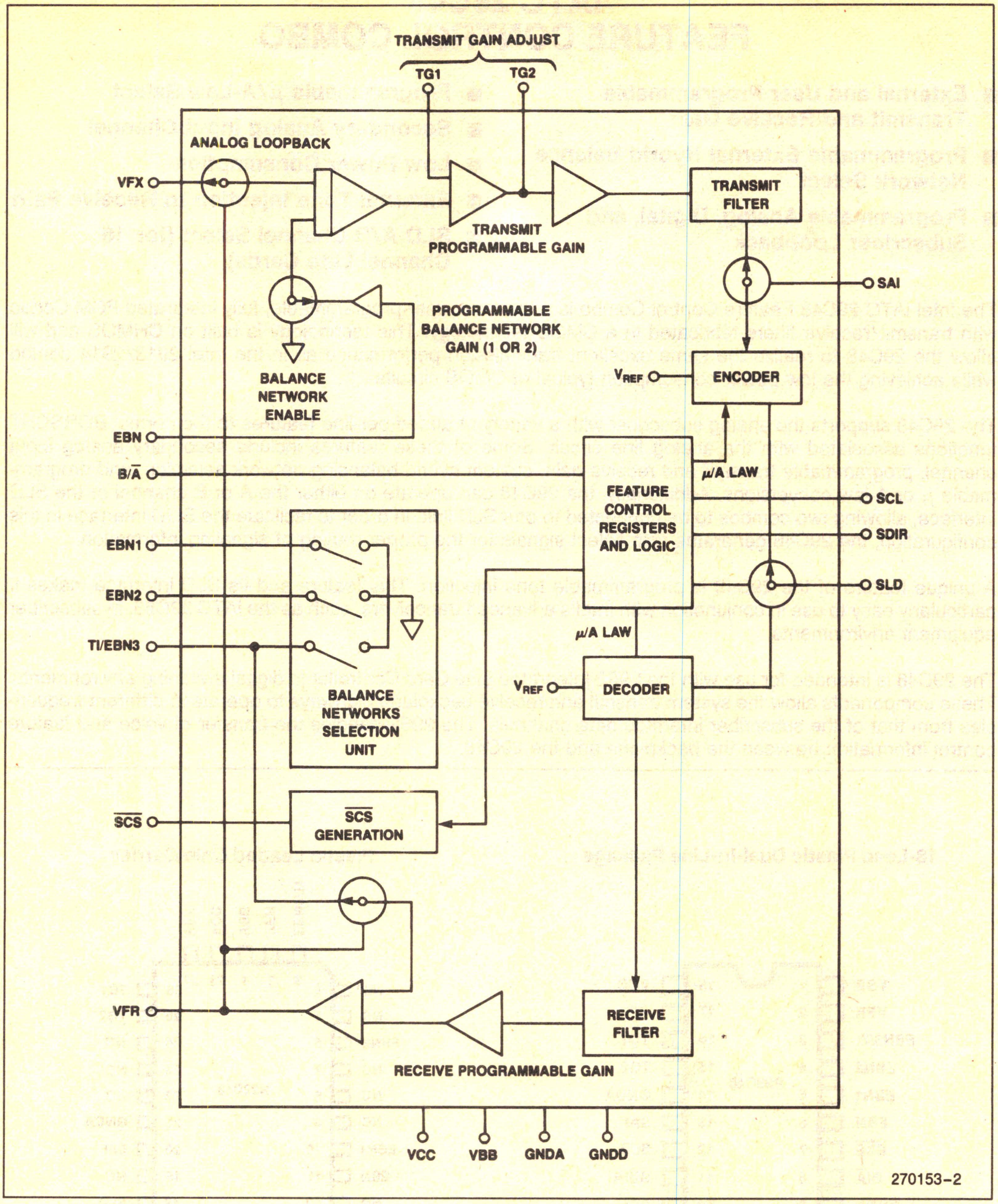


Figure 2. Block Diagram

Table 1. Pin Names

VFX	Analog Input	SCL	Subscriber Clock
VFR	Analog Output	SLD	Subscriber Data Link
GNDD	Digital Ground	SDIR	Subscriber Direction
GNDA	Analog Ground	TG1, TG2	Transmit Gain Adjust
VCC	Power (+5V)	EBN1/2	External Balance Network Selection Inputs
VBB	Power (−5V)	EBN3/TI	External Balance Network Selection Input Or Tone Injection
B/$\overline{A}$	Channel Selection	EBN	External Balance Input
$\overline{SCS}$	SLIC Chip Select	SAI	Secondary Analog Input

Table 2. Pin Description

Symbol	Function
VCC	Most positive supply; input voltage is +5V ±5%.
VBB	Most negative supply; input voltage is −5V ±5%.
GNDA	Analog ground return line. Not internally connected to GNDD.
GNDD	Digital ground return line. Not internally connected to GNDA.
VFX	Analog voice input to transmit channel. Input impedance is typically larger than 100 KΩ.
TG1	Inverting input to transmit gain adjusting op-amp. Feedback point for external gain adjusting resistor network or frequency compensation network. Input impedance is typically larger than 10 MΩ.
TG2	Output of the transmit gain adjusting op-amp. Will drive external gain adjusting resistor network as well as frequency compensation network with impedance at least larger than 10 KΩ.
VFR	Receive voice output. Capable of directly driving transformer hybrids or impedance loads of 300 Ω to ±3.2 Vp.
EBN	Input to the hybrid balancing circuit. Input impedance is typically larger than 10 MΩ.
EBN1	Input connected to a grounded switch. The switch's on resistance is not greater than 600 Ω.
EBN2	Input connected to a grounded switch. The switch's on resistance is not greater than 600 Ω.
EBN3/T1	This pin is multiplexed according to the feature control registers. When programmed to be EBN3, it is an input connected to a grounded switch. The switch's on resistance is not greater than 600 Ω. If this pin is programmed to be TI, an analog signal applied on this pin will be added to the received voice signal before the receive power amplifier.
SCL	Subscriber clock. Supplied by the line card controller, this is a 512 KHz, 50% or 33% duty cycle clock. Input will accept TTL levels.
SDIR	Subscriber direction signal and frame sync. When high, SLD becomes an input and data is transferred from the line card controller to the 29C48. When low, the output buffer on the 29C48 SLD pin is enabled and data is transferred from the 29C48 to the controller. Input will accept TTL levels.
SLD	Subscriber Line Datalink. A 512 Kbps bi-directional serial data port, which is clocked by SCL. SLD becomes a TTL compatible input when SDIR is high and an output capable of driving one TTL load when SDIR is low, during the appropriate SLD fields for the assigned channel.
B/$\overline{A}$	Pin strapped to assign the 29C48 to process either A or B channel information from the SLD bus. A low level (GNDD) on this pin selects channel A, a high level (VCC) channel B.
$\overline{SCS}$	This pin is a TTL compatible output capable of driving one TTL load: when low, it informs a SLIC device connected to the same SLD bus as the 29C48 that it can process the receive and transmit signalling data of the present SLD frame.
SAI	Secondary analog input.

FUNCTIONAL DESCRIPTION

The 29C48 is a combined channel filter and PCM codec for use on analog line interface circuit boards in a digital telecommunications switching system. This device resides between the circuitry which provides the "BORSHT" functions for a given line, and the shared line board controller. It provides the transmit and receive voice-path filtering and companded analog-to-digital and digital-to-analog conversions necessary to interface a full duplex (4-wire) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. (See Figures 3a and 3b for typical line card applications.)

The 29C48 incorporates additional features making it particularly suited to subscriber applications. Tone injection allows easy implementation of DTMF feedback and side tone injection, and secondary analog signal input allows remote control and monitoring. (See Figure 4 for a typical ISDN subscriber equipment application.)

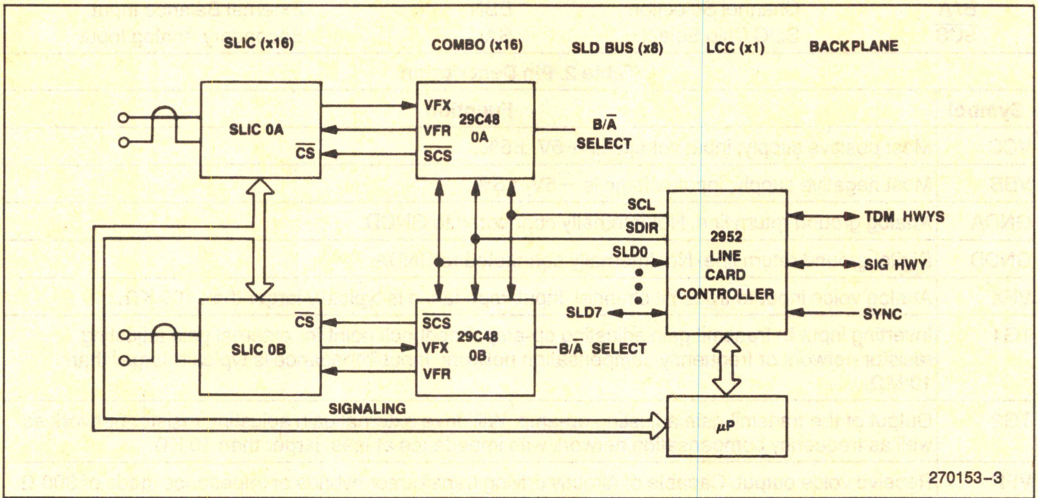


Figure 3a. Analog Line Card with Discrete or Electronic Parallel Control SLICs

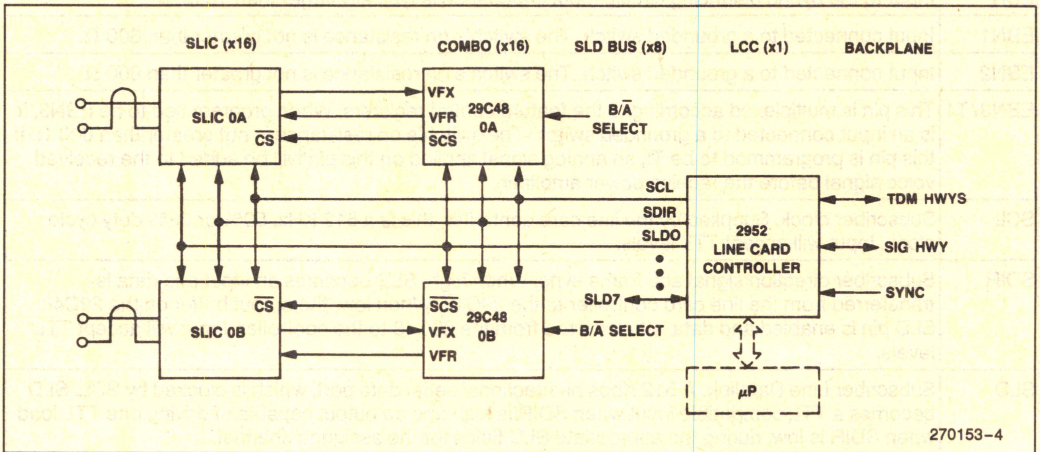


Figure 3b. Analog Line Card with SLD Compatible SLICs

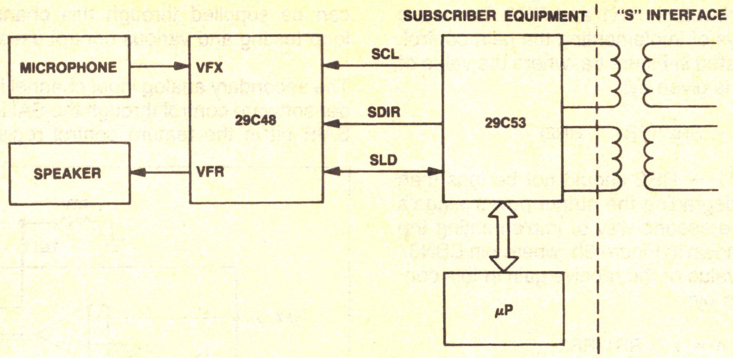


Figure 4. Subscriber Equipment

TRANSMIT AND RECEIVE OPERATION

Transmit Filter

A low pass anti-aliasing section is included on chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. The 29C48 specifications meet the digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 12.

A high pass section configuration rejects low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Gain of up to 20 dB can be set without degrading the performance of the filter. The transmit filter also provides additional loss at 12 KHz and 16 KHz (frequencies of metering pulses).

Encoding

The output of the transmit filter or the secondary analog input is internally sampled by the encoder and held on an internal sample and hold capacitor. DC offset is corrected by an on-chip auto zero circuit. The signal is then encoded and presented as PCM data on the SLD lead. (First or second byte of the transmit half-frames depending upon the channel assignment of the device.)

Decoding

The PCM word received on the SLD lead (first or second byte of the receive half-frame, depending upon the channel assignment of the device) is sent to the decoder after a serial to parallel conversion. The decoded value is held on an internal sample and hold capacitor.

Receive Filter

The receive section of the filter provides a passband flatness and stopband rejection which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. It also provides additional loss at 12 KHz and 16 KHz. The receive filter transfer characteristics and specifications will be within the limits shown in Figure 13.

GENERAL OPERATION

External Gain Setting

Both transmit and receive gain levels are factory trimmed, but can be modified by external resistors during line card assembly. The value of transmit gain is adjusted by connecting resistors RT1 and RT2 (see Figure 5) at the two external gain setting control pins, TG1 and TG2. These two pins are the input and output of an on-board gain amplifier stage, and the resistors provide the necessary input and feedback for gain control. The value of external gain is given by:

$$A = 1 + RT1/RT2$$

For unity gain, pins TG1 and TG2 are tied together.

For the receive section, the external gain can be set by the external resistors, RR1 and RR2. There are two possible ways of implementing the gain control. The first is illustrated in Figure 6a, where the value of the receive gain is given by:

$$A = RR2/(RR1 + RR2)$$

The value of RR1 + RR2 should not be less than 600 Ω to avoid degrading the output power stage's performance. The second way of implementing the receive gain is shown in Figure 6b, where pin EBN3/T1 is used. The value of the receive gain in this configuration is given by:

$$A = 1 + RR1/RR2$$

Hybrid Balancing Network

Three external balancing networks can be applied to the 29C48 by the user to accommodate varying subscriber loop characteristics (see Figure 7 for external connections). Feature control allows the grounding of any combination of these networks in order to best suit a particular application. Feature control also allows the user to select a gain of 0.0 or +6.0 dB in the balance signal path to suit the type of SLIC used.

FREQUENCY COMPENSATION

The user may, if desired, compensate for the frequency response characteristics of the SLIC by adjusting the frequency response of the transmission chain. This can be accomplished in the same way as the external gain setting is done in the transmit and receive directions. But, instead of using purely resistive impedances, resistor and capacitor networks have to be used to achieve complex impedances. The two compensation schemes are shown in Figures 8a and 8b. The gains in the transmit and receive directions are respectively:

for Figure 8a $A = 1 + ZT1/ZT2$

for Figure 8b $A = 1 + ZR1/ZR2$

SECONDARY ANALOG INPUT

Although the main application of the 29C48 will be for voice transmission, it also offers a secondary unfiltered input channel. Narrow band analog signals can be supplied through this channel for remote loop testing and various control uses.

The secondary analog input channel is accessed under software control through the SAI input. When the SAIE bit in the feature control register is set to a

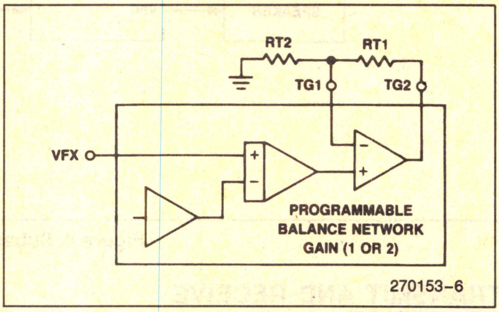


Figure 5. Transmit Gain Setting

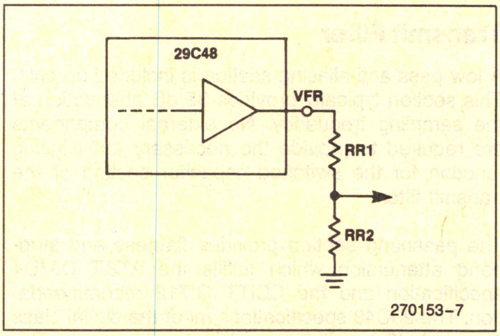


Figure 6a. Receive Gain Setting

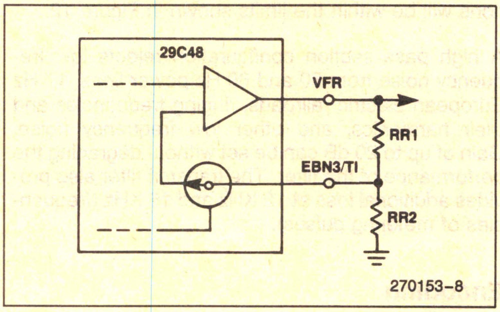


Figure 6b. Receive Gain Setting

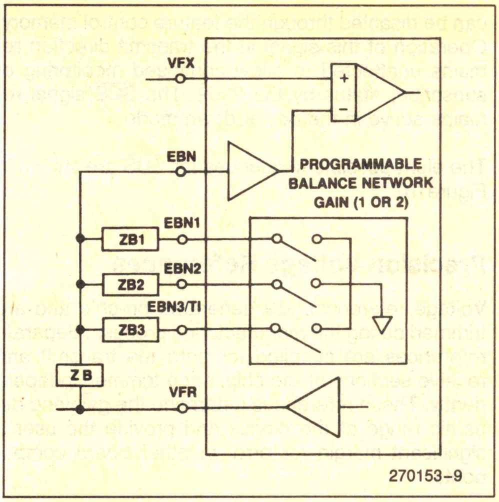


Figure 7. Balance Networks

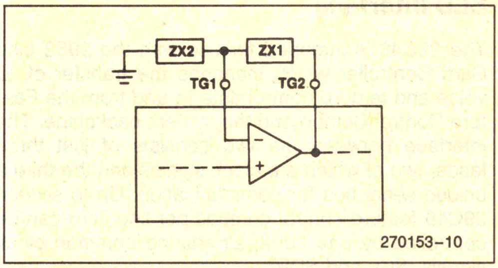


Figure 8a. Transmit Frequency Compensation

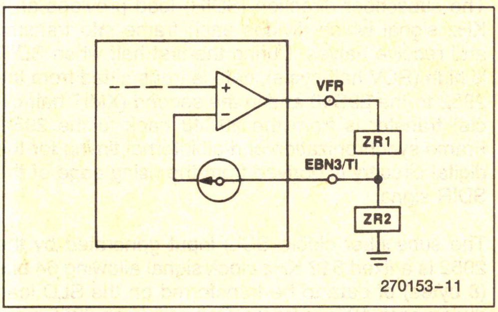


Figure 8b. Receive Frequency Compensation

logical one, the 29C48 will encode and transmit the signal present at the SAI input. The 29C48 will switch back to transmission of the voice signal as soon as the SAIE bit is set back to a logical zero.

TONE INJECTION

When specified by the feature control memory, an audio frequency signal applied to the EBN3/T1 pin will be added to the receive voice signal at the power amplifier. This feature allows easy implementation of DTMF feedback and side tone injection in digital telephone applications, as well as injection of call waiting or metering tones in line card applications. A typical application is shown in Figure 9. Here VFR is the combination of the receive voice signal (V0) and two tones (V1 and V2).

$$VFR = 2V0 + (V1 + V2)/2$$

CHANNEL ASSIGNMENT

Two 29C48s can be attached to the same SLD line to exchange information with the line card controller during each SLD frame.

The B/$\overline{A}$ pin of the 29C48 is used to assign a voice channel of the SLD frame to the device. When the B/$\overline{A}$ pin is tied low, the 29C48 operates as an A-channel combo, receiving and transmitting voice during the first and fifth bytes of the SLD frame. When this pin is tied high, the 29C48 operates as a B-channel combo, receiving and transmitting voice during the second and sixth bytes of the SLD frame.

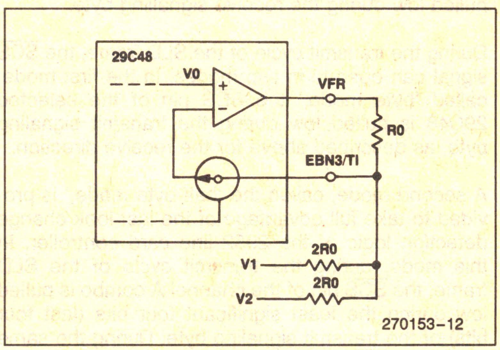


Figure 9. External Tone Injection

The feature control receive and transmit channels of the SLD frame are shared by the two 29C48s. A 29C48 will accept or return feature control information only if it has been instructed to do so during the first byte of a feature control frame. This is accomplished by setting the logic level of the channel selection bit in feature control byte #1 to match the logic level of the B/$\overline{\text{A}}$ pin of the appropriate 29C48. The selected 29C48 will keep exchanging feature control information with the line card controller until a new framing byte makes a new selection. The status of the channel select bit is sent back to the line card controller during the seventh byte of each SLD frame, making it possible to determine which channel is transmitting feature control information.

The 29C48 does not process data received in the signaling channel. However, it generates chip select signals during the appropriate time slots in order to facilitate the SLIC interface. (See section on SLIC Chip Select.) The 29C48 enters into a high impedance state during the signaling transmit channel, the eighth byte of the SLD frame.

SLIC Chip Select

In order to facilitate interfacing to an SLD compatible SLIC, especially when two SLICs share the same SLD line, the 29C48 includes a programmable chip select signal.

During the receive cycle of the SLD frame, the $\overline{\text{SCS}}$ pin of the 29C48 whose channel selection pin (B/$\overline{\text{A}}$) has the same logic state as the channel selection bit (see previous section on Channel Assignment) is pulled low during the receive signalling byte.

During the transmit cycle of the SLD frame, the $\overline{\text{SCS}}$ signal can operate in two modes. In the first mode, called 'byte mode,' the $\overline{\text{SCS}}$ pin of the selected 29C48 is pulled low during the transmit signaling byte, as described above for the receive direction.

A second mode, called the 'half-byte mode,' is provided to take full advantage of the last look change detection logic of the 2952 line card controller. In this mode, during the transmit cycle of the SLD frame, the $\overline{\text{SCS}}$ pin of the channel A combo is pulled low during the least significant four bits (last four bits) of the transmit signaling byte. During the same frame, the $\overline{\text{SCS}}$ pin of the channel B combo is pulled low during the most significant four bits (first four bits) of the transmit signaling byte. This allows signaling data from both A and B channel SLD compatible SLICs to be processed by the 2952 during the same frame.

To minimize power consumption, operation of the $\overline{\text{SCS}}$ signal during the receive half of the SLD frame can be disabled through the feature control memory. Operation of this signal in the transmit direction remains unaffected to allow continued monitoring of subscriber status by the 2952. The $\overline{\text{SCS}}$ signal remains active in the power down mode.

The eight possible sequences for $\overline{\text{SCS}}$ are shown in Figure 10.

Precision Voltage References

Voltage references are generated on-chip and are trimmed during the manufacturing process. Separate references are supplied for both the transmit and receive sections of the chip, each trimmed independently. These references determine the gain and dynamic range of the device and provide the user a significant margin for error in other board components.

SLD Interface

The 29C48 is intended for use with the 2952 Line Card Controller which manages the transfer of all voice and feature control data to and from the Feature Control Combo and the system backplane. The interface between the two consists of just three leads, two of which are clock signals and the third a unique serial bus for communication. Up to sixteen 29C48 feature control combos per line card can be controlled by one 2952, all sharing common clock signals, SCL and SDIR.

The subscriber direction (SDIR) lead provides an 8 KHz signal which divides each frame into transmit and receive halves. During the first half when SDIR is high (RCV half-cycle), data is transmitted from the 2952 to the 29C48 and in the second (XMIT half-cycle) transfer is from the 29C48 back to the 2952. Frame synchronization and all internal timing for the digital circuitry is derived from the rising edge of the SDIR signal.

The subscriber clock (SCL) input generated by the 2952 is a fixed 512 KHz clock signal allowing 64 bits (8 bytes) of data to be transferred on the SLD lead during each 125 μs frame. Depending on 2952 master clock frequency, the SCL duty cycle can be either 50% or 33%.

The Subscriber Line Datalink (SLD) is a bi-directional serial bus that transfers four bytes of serial data to and from the 29C48 each frame. During the first half of each frame, RCV channel information is expected by the 29C48 as two bytes consisting of voice and feature control information, while the other two bytes of the RCV half-frame are simply ignored. Similarly, during the second half-frame, one byte of voice and,

if so instructed by the line card controller, one byte of feature control information is sent by the 29C48. The 29C48 places its SLD lead in a high impedance state while the other device connected to the SLD line transmits its own information. The most significant bit (bit 7) of each byte is sent first on the SLD line. The data format of an SLD frame is shown in Figure 11.

Upon power supply application, feature control read or write of the 29C48 is disabled for 9 SLD frames. During this time, the 29C48 resets and enters the power down mode. The $\overline{SCS}$ output remains high until the 29C48 has been configured as an A or B channel device by the first write to feature control byte #1.

PROGRAMMABLE FEATURES

The 29C48 is configured by the 2952 line card controller by a set of five feature control bytes (FCB).

Receive $\overline{SCS}$ Disabled (See Section on SLIC Chip Select)

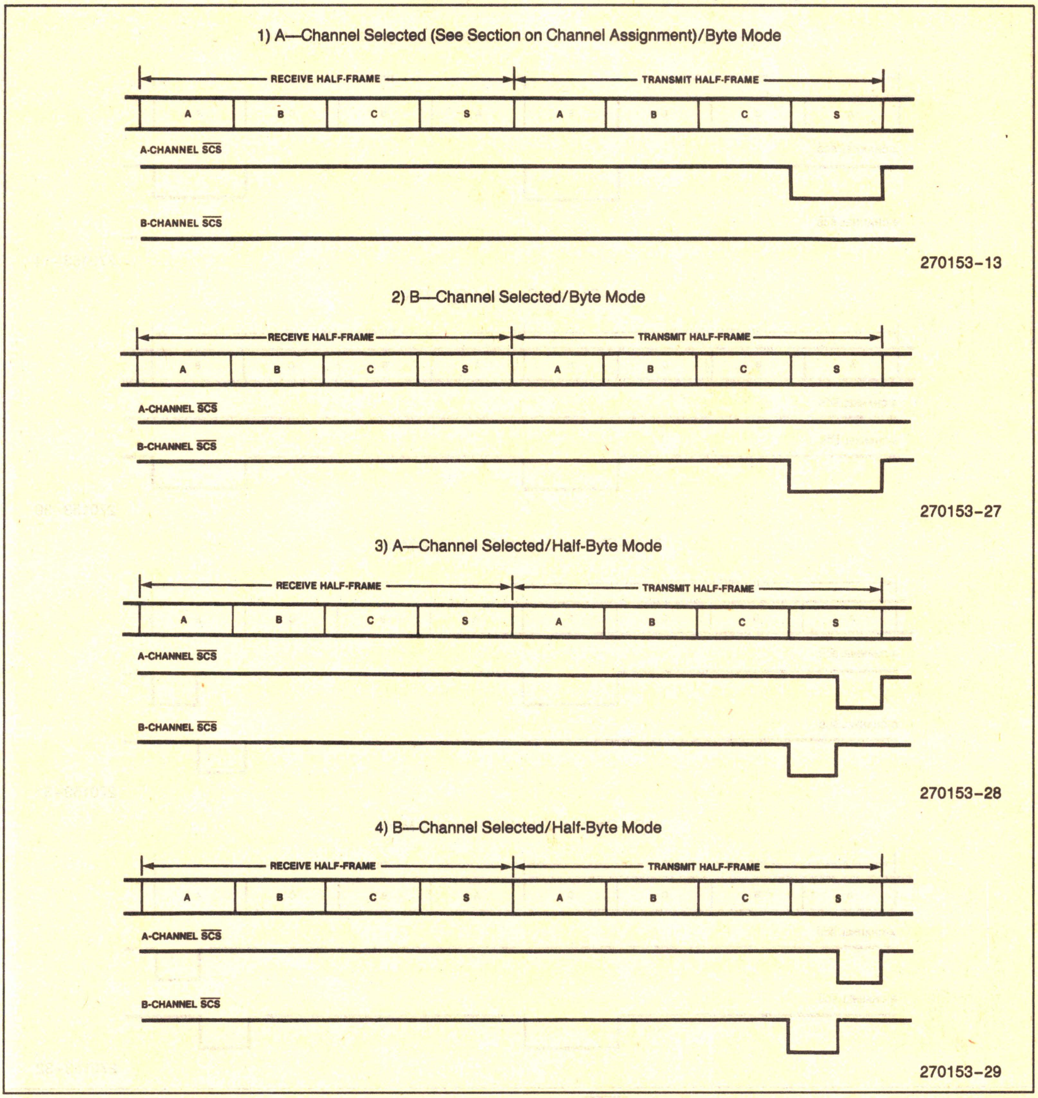


Figure 10. $\overline{SCS}$ Timing Diagram

These bytes of information are stored in internal registers which are serially multiplexed to and from the SLD interface in the third and seventh byte locations. The first two bits of each byte consist of a multiframe synchronization and write enable code. The framing bit (bit 7, MSB) establishes the beginning of a feature control frame when set to a logical zero, and increments the feature control counter when set to one. The second (bit 6) enables the writing to the 29C48 when it is the logical complement of the framing bit. In addition to the two header bits, feature control byte #1 also includes a channel selection bit (bit 0, LSB). This bit is used to designate one of the two 29C48s sharing an SLD link for feature control information exchange. (See previous section on channel Assignment.)

When writing new feature control information to the 29C48, the first byte should contain a framing (F) and write enable (WE) header of 01 (F=0 and WE=1), and an appropriate channel selection bit. This designates a new frame of information to trans-

Receive $\overline{\text{SCS}}$ Enabled

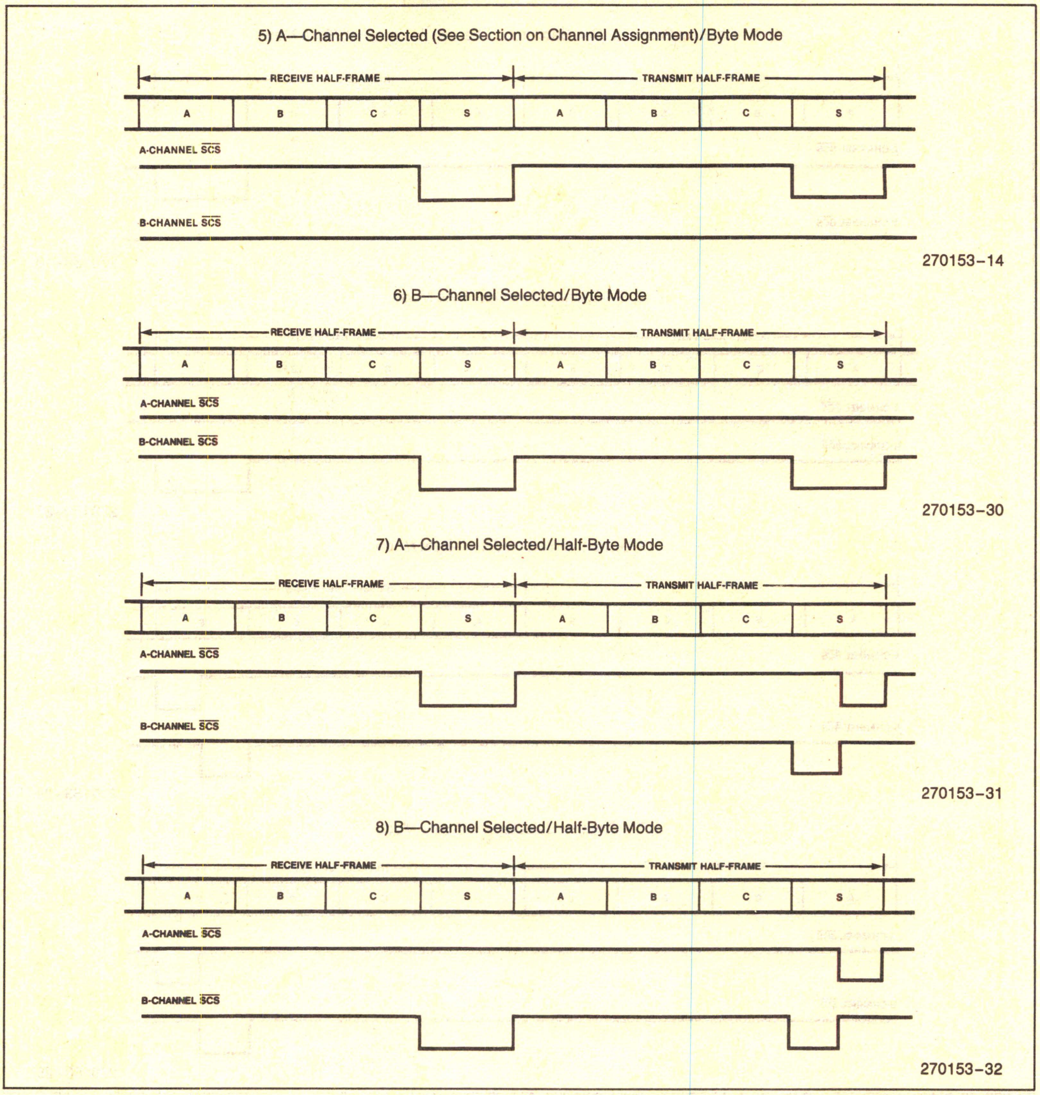
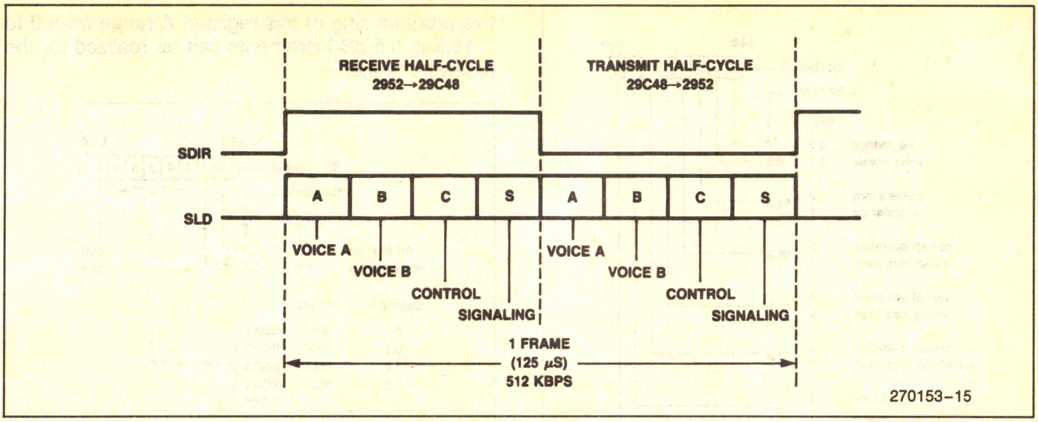


Figure 10. $\overline{\text{SCS}}$ Timing Diagram (Continued)



Voice A, Voice B: A and B channel voice bytes respectively.
Control: Feature control information. This information is exchanged with the 29C48 whose channel selection pin matches the channel selection bit of the latest framing feature control byte.
Signaling: Signaling information which controls the subscriber line. The 29C48 enters into a high impedance state during this time slot, and generates a chip select signal (see section on SLIC chip select).

Figure 11. 29C48/2952 Interface

fer. The subsequent bytes should each have F = 1 to advance the counter, and WE = 0 to enable the write operation.

The controller can also request to verify the feature control register contents by sending a 00 or 11 at the beginning of the byte to be read. To read the first byte, a 00 F/WE code and an appropriate channel selection bit should be sent while each subsequent byte should have a 11 header. An internal six-stage counter is set on the first byte verified then incremented once each 125 μs frame. It is reset only upon detection of a 01 or 00 F/WE. Once the counter is greater than five, neither read nor write modes may be selected by sending the 29C48 a 11 or 10 framing and write enable code. While in this state, the 29C48 will then echo in byte 7 the data it received in byte 3. Another feature control information exchange cycle can only be initiated by establishing a new feature control frame (sending F = 0).

FCB #1—Power Up/Down, Loop Back Mode, μA/-Law, Channel Select Register

POWER UP AND DOWN

The 29C48 can be instructed to go into the power down or standby mode for reduced power consumption. In this mode, all analog inputs and outputs are placed in a high impedance state, inhibiting voice signals. A code of all ones will be output in the voice byte on the SLD. Signaling and feature control information will continue to be processed to allow the 29C48 to be read or reprogrammed.

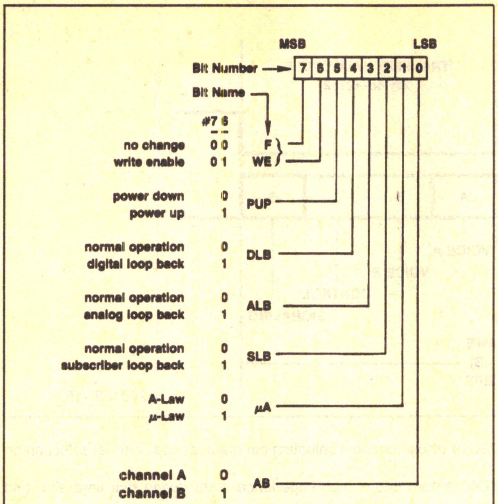
The 2952 can change the state of the feature control combo from standby to active by sending the first feature control byte only. All other register contents will be preseved during power down provided the power supplies remain connected.

LOOP BACK MODE SELECT

Three modes of remote testing are incorporated in the 29C48 and can be selected by appropriate coding in this register. The loopback features allow a number of tests to be performed to determine line quality and balancing. These include digital loop back, analog loop back, and subscriber loop back.

In the digital loopback mode, the combo retransmits the PCM word it receives in the voice A or B byte of the SLD back to the line card controller in the same frame. This feature allows path verification and testing of the circuit up to the combo.

When the analog loopback mode is selected, the analog output VFR is internally connected to the analog input VFX. This feature allows functional testing of the combo as well as gain adjustment.



270153-16

In the third test mode, subscriber loopback, the digital output of the A/D converter is internally connected to the input of the D/A converter. The analog signal input to VFX is sent through the transmit filter, encoded, then decoded, filtered and output to VFR. This mode is used primarily for simplifying analog to analog testing from the subscriber side of the line card. Simultaneous selection of more than one loopback mode is prohibited.

CONVERSION LAWS

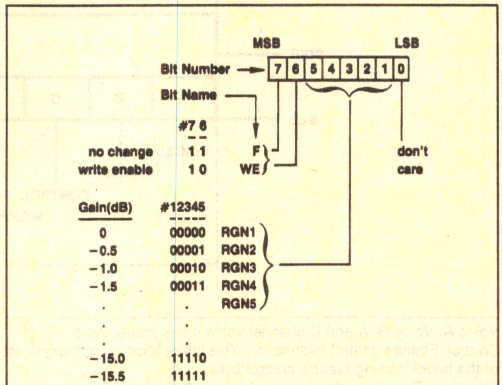
The 29C48 can be selected for either μ-law or A-law operation. A user can select either conversion law by assigning the corresponding bit. A logical 1 in bit 1 would select μ-law while a logical 0 would select A-law conversions. Both conversions follow CCITT recommendation G.711.

FEATURE CONTROL EXCHANGE CHANNEL SELECT

The LSB of feature control byte #1 is the channel selection bit. It is used to select one of the two 29C48s sharing an SLD line for feature control information exchange. A logical zero will select the channel A combo, and a logical one will select the channel B combo.

FCB #2—Receive Programmable Gain Register

The receive gain levels can be adjusted by applying external resistors as mentioned earlier, or by selective programming of this register. A range from 0 to −15.5 in 0.5 dB increments can be realized for the receive channel.



270153-17

FCB #3—Secondary Analog Channel, Chip Select, and Tone Injection Register

SECONDARY ANALOG INPUT

The 29C48 can be instructed to switch the input of its encoder to the secondary analog input by setting the SAIE bit to a logical one. Transmission of the voice signal will resume as soon as SAIE is set back to a logical zero.

PROGRAMMABLE SLIC CHIP SELECT

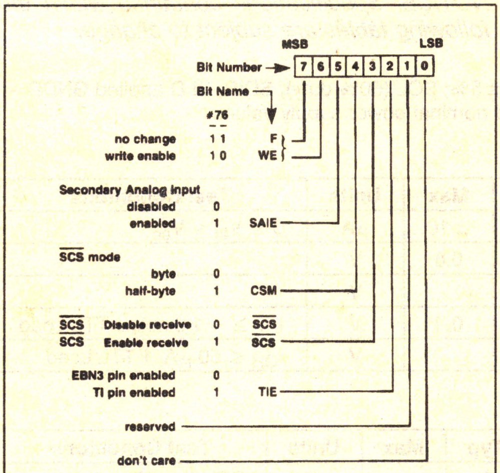
Although the 29C48 does not process signaling information, it generates chip select signals in order to help in interfacing to SLD compatible SLICs.

During the transmit half-frame, the chip select works in two possible modes determined by the CSM bit. In the byte mode, the $\overline{SCS}$ pin of the 29C48 selected by the chip select bit in feature control byte #1, will be pulled low during the XSIG byte. In the half-byte mode, the $\overline{SCS}$ pin of the A-channel 29C48 will be pulled low during the four least significant bits of the XSIG byte, and the $\overline{SCS}$ pin of the B-channel 29C48 will be pulled low during the four most significant bits of the XSIG byte.

Generation of chip select signals during the receive half frame can be disabled by setting the CSD bit to a logical zero.

TONE INJECTION

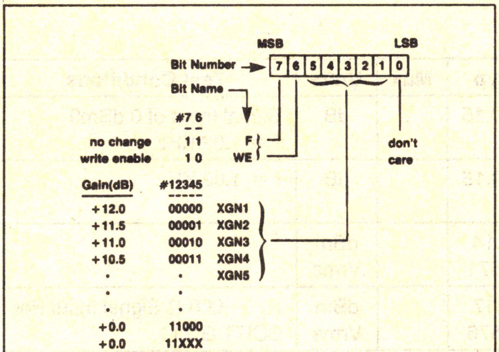
When the TIE bit is set to a logical one, audio signal applied at the EBN3/T1 pin will be added to the output of the receive programmable gain module. This feature can be used for easy implementation of side tone injection and DTMF feedback, as well as injection at the line card of call waiting tones, ringing or metering pulses.



270153-18

FCB #4—Transmit Programmable Gain Register

The gain setting of the transmit section of the chip operates in the same manner as the receive gain register. A 12 dB range from 0.0 dB to +12.0 dB in 0.5 dB increments is available.



270153-19

FCB #5—Balance Network Select and Gain Register

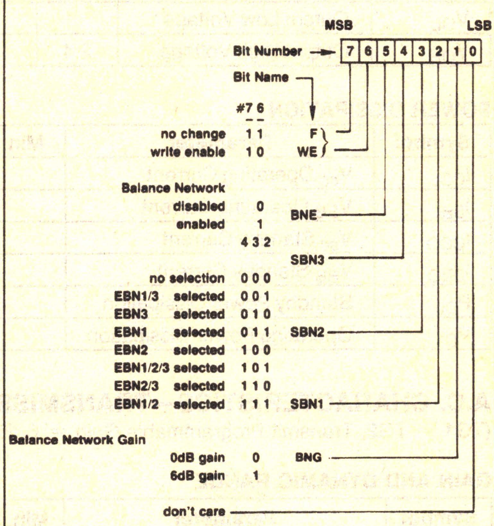
BALANCE NETWORKS

Three external balance networks can be used with the 29C48. Feature control allows the selection of network EBN1, EBN2, and EBN3 individually or in combination in order to best suit a particular application.

EBN3 selection is not effective when TIE is set to a logical one.

GAIN SETTING

An additional 6 dB gain in the balance signal path can be realized by coding this bit to a logical one. A logical zero provides unity gain.



270153-20

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias ....−10°C to +80°C
Storage Temperature ....−65°C to +150°C
All Input and Output Voltages
with Respect to V_{BB}..............−0.3V to 13V
All Input and Output Voltages
with Respect to V_{CC}..............−13V to 0.3V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%; SCL (50% duty), SDIR, SLD applied GNDD = 0V, GNDA = 0V. Typical values are for T_A = 25°C and nominal power supply values

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current			±10	μA	$0 \leq V_{in} \leq V_{cc}$
V_{IL}	Input Low Voltage			0.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	$I_{OL} \geq -1.6$ mA, 1 TTL Load
V_{OH}	Output High Voltage	2.4			V	$I_{OH} \leq 50$ μA, 1 TTL Load

POWER DISSIPATION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{CCL}	V_{cc} Operating Current		9		mA	RCV Signaling Byte = 0
I_{BBL}	V_{BB} Operating Current		9		mA	RCV Signaling Byte = 0
I_{CCO}	V_{cc} Standby Current		0.8		mA	
I_{BBO}	V_{BB} Standby Current		0.8		mA	
P_{D0}	Standby Power Dissipation		8		mW	
P_{D1}	Operating Power Dissipation		90		mW	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

(TG1 = TG2, Transmit Programmable Gain = 6 dB. Receive Programmable Gain = 0 dB)

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response Tolerance		±0.15		dB	Signal Input of 0 dBm0 f = 1.02 KHz
DmW	Digital Milliwatt Response Tolerance		±0.15		dB	f = 1.02 KHz
$DmW_{\mu V}$	Digital Milliwatt Response VFR, μ-law		6.14 1.571		dBm Vrms	
DmW_{AV}	Digital Milliwatt Response VFR, A-law		6.17 1.576		dBm Vrms	R_L = 600 Ω Signal Input Per CCITT G.711
$OTLP_{\mu X}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.12 0.785		dBm Vrms	μ-law Referenced to 600Ω
$OTLP_{AX}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.15 0.788		dBm Vrms	A-law Referenced to 600Ω

GAIN TRACKING

Reference level = 0 dBm0 at 1.02 KHz, TG1 = TG2, Transmit Programmable Gain = 6 dB, Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
GT_T	Transmit Gain Tracking Error Sinusoidal Input; μ or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
GT_R	Receive Gain Tracking Error Sinusoidal Input; μ or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 AT&T PUB43801 and CCITT G.712—Method 2

ANALOG INTERFACE, RECEIVE CHANNEL

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
R_{OR}	Output Resistance, VFR		1		Ω	
V_{OSR1}	Output Offset, VFR		50		mV	Relative to GNDA
C_{LR}	Load Capacitance, VFR			100	pF	
V_{OR1}	Max Output Voltage Swing Across R_L, VFR	±3.2			V_p	$R_L \geq 300\ \Omega$

ANALOG INTERFACE, TRANSMIT PRIMARY AND SECONDARY CHANNELS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{BX}	Input Leakage Current, EBN, TG1, T1		100		nA	−1.6V < VFX < 1.6V
R_{IX1}	Input Resistance, VFX		100		KΩ	−1.6V < VFX < 1.6V
R_{IX1}	Input Resistance, EBN, TG1, T1		10		MΩ	±1.6V < VFX < 1.6V
TGmax	Max Transmit Gain Adjust			20	dB	
V_{OTG}	Max Output Voltage Swing TG2			±1.6	V	$R_L \geq 10K\ \Omega$(1)
C_{LX}	Load Capacitance, TG2			20	pF	
R_{LX}	Load Resistance, TG2	10			KΩ	
R_{GND}	On Resistance to GND, EBN1, EBN2, ENB3			600	Ω	

NOTE:
1. Maximum output voltage swing at TG2 should be less than ±1.6V in order to avoid clipping in later stages.

DISTORTION (Primary Channel)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
SD_X, SD_R	Signal to Distortion, μ or A-law Sinusoidal Input; CCITT G.712—Method 2 Half Channel	35 29 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X, DP_R	Single Frequency Distortion Products in Band (2nd or 3rd Harmonic Half Channel)		−50	−46	dB	Input = 1.02 KHz 0 dBm0 AT&T Advisory #64 (3.8)
IMD_1	Intermodulation Distortion, End to End Measurement			−40	dBm0	CCITT G.712(7.1)
IMD	Intermodulation Distortion, End to End Measurement			−50	dBm0	CCITT G.712(7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−27	dBm0	CCITT G.712(6.1)
SIS	Spurious in Band Signals, End to End Measurement			−40	dBm0	CCITT G.712(9)
D_{AX}	Transmit Absolute Delay		180		μs	0 dBm0, 1.02 KHz Includes Delay Through A/D
D_{DX}	Transmit Differential Envelope Delay; Relative to Minimum Envelope Delay (1.4 KHz)		170 95 45 105		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz
D_{AR}	Receive Absolute Delay		125		μs	0 dBm0, 1.02 KHz Includes Delay Through D/A
D_{DR}	Receive Differential Envelope Delay; Relative to Minimum Envelope Delay (300 Hz)		45 35 85 110		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz

NOISE (Primary Channel)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted		12		dBmC0	Transmit Gain Adjust = 0 dB
N_{XP1}	Transmit Noise, Psophometrically Weighted		−78		dBm0p	Transmit Gain Adjust = 0 dB
N_{RC1}	Receive Noise, C-Message Weighted		10		dBmC0	Unity Gain; Idle Code
N_{RP1}	Receive Noise, Psophometrically Weighted		−80		dBm0p	Unity Gain; Idle Code
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit or Receive Channel		−35		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; Note 1.
$PSRR_2$	V_{BB} Power Supply Rejection Transmit or Receive Channel		−30		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; Note 1.

NOTE:
1. Measured at SLD Voice bytes for transmit channel. Measured at V_{FR} for receive channel.

CROSSTALK

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
CT_{TR}	Crosstalk, Transmit Voice to Receive Voice		−75		dB	Input = 0 dBm0, Unity Gain 1.02 kHz; Idle Code on SLD Voice Byte
CT_{RT}	Crosstalk, Receive Voice to Transmit Voice		−75		dB	0 dBm0, 1.02 KHz Signal at SLD Receive Voice Byte; VFX = GNDA

TRANSMIT VOICE FREQUENCY CHARACTERISTICS

TG1 = TG2, Transmit Programmable Gain = 6 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at VFX
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−22	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3400 Hz	−0.70		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	
ΔG_{PX}	Programmable Gain		±0.10		dB	Freq. = 1.02 KHz for All Steps

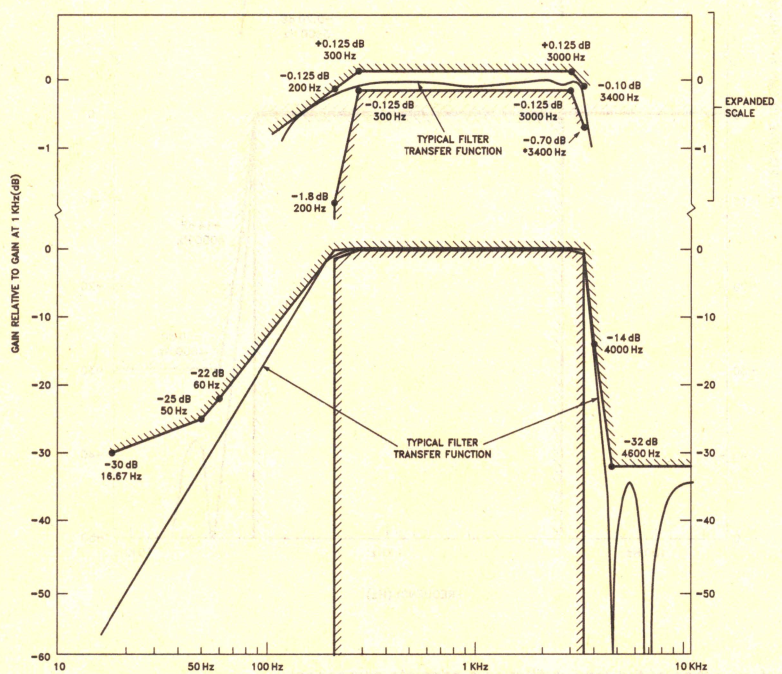
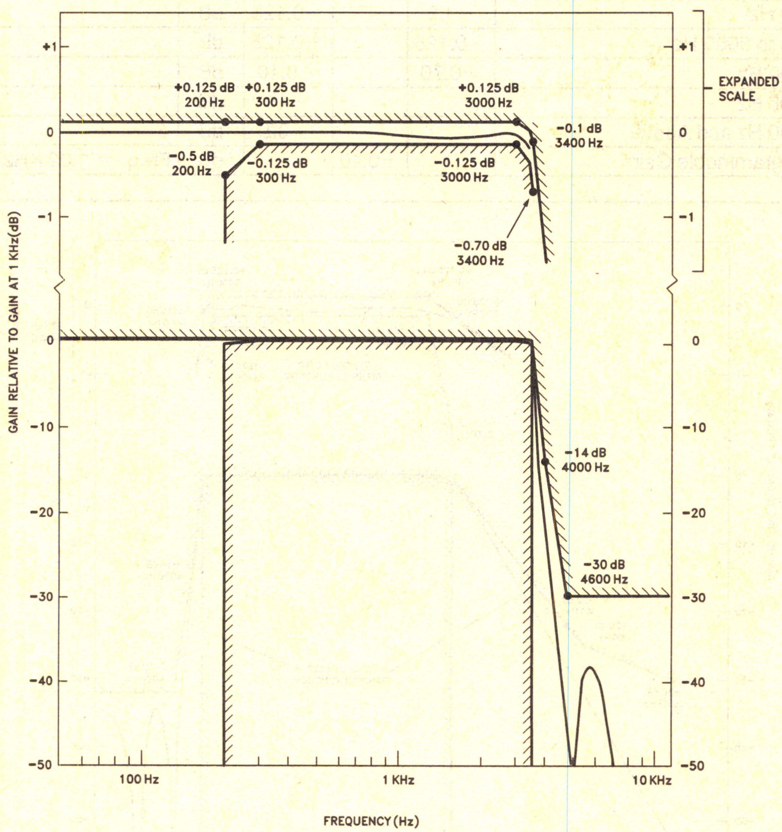


Figure 12. Transmit Voice Frequency Characteristics

RECEIVE VOICE FREQUENCY CHARACTERISTICS

Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Input on SLD
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3400 Hz	−0.70		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz & Above			−30	dB	
ΔG_{PR}	Programmable Gain Accuracy		±0.10		dB	f = 1.02 KHz All Steps



270153–22

NOTES:
1. Typical transfer function of the receive filter as a separate component.
2. Typical transfer function of the receive filter driven by the sample and hold output of the Intel 2910A and 2911A codecs. The combined filter/codec response meets the stated specifications.

Figure 13. Receive Voice Frequency Characteristics

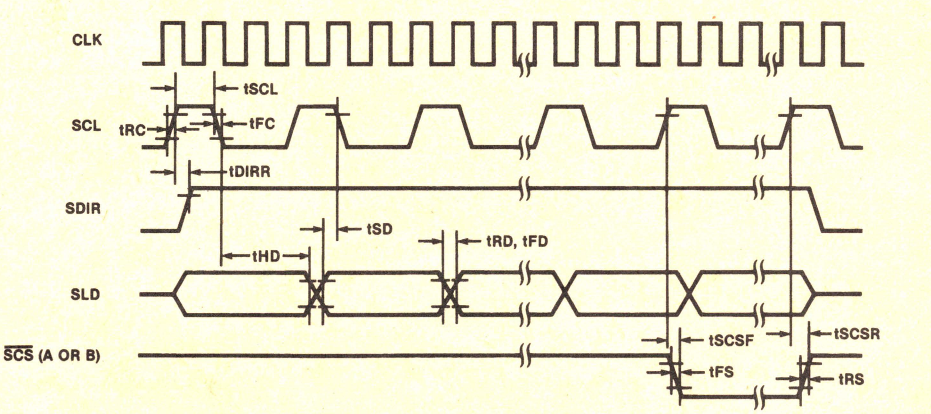
A.C. CHARACTERISTICS—TIMING PARAMETERS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
T_{SCL}	SCL Pulse Width	550		1075	ns	
T_{DC}	SCL Duty Cycle	28		55	%	
T_{RC} T_{FC}	Rise, Fall Times, SCL			50	ns	
T_{RD} T_{FD}	Rise, Fall Times, SLD			50	ns	
T_{RS} T_{FS}	Rise, Fall Times, $\overline{SCS}$			50	ns	50 pF Load
T_{DIRR}	SCL to SDIR Delay	−150		150	ns	Receive Cycle
T_{DIRF}	SCL to SDIR Delay	−150		420	ns	Transmit Cycle
T_{DD}	SCL to SLD Delay	0		200	ns	29C48 Transmitting
T_{SD}	Set-up Time, SLD to SCL	100			ns	2952 Transmitting
T_{HD}	Hold Time, SCL to SLD	100			ns	
T_{HZ1}	SDIR to SLD Active	0		100	ns	Byte 1, Bit 1 29C48 Transmitting, Channel A
T_{HZ2}	SCL to SLD High Impedance	0		100	ns	After Last Bit of Channel A, B, or Feature Control as Appropriate (Channel A/B Operation)
T_{HZ3}	SCL to SLD Active	0		100	ns	Channel A or B, Feature Control, Signaling as Appropriate (Channel A/B Operation)
T_{SCSF}	SCL to $\overline{SCS}$ Low	0		200	ns	50 pF Load
T_{SCSR}	SCL to $\overline{SCS}$ High	0		200	ns	50 pF Load

*In cases where T_{DIRF} is positive, T_{DD} is to be measured from the SDIR edge.

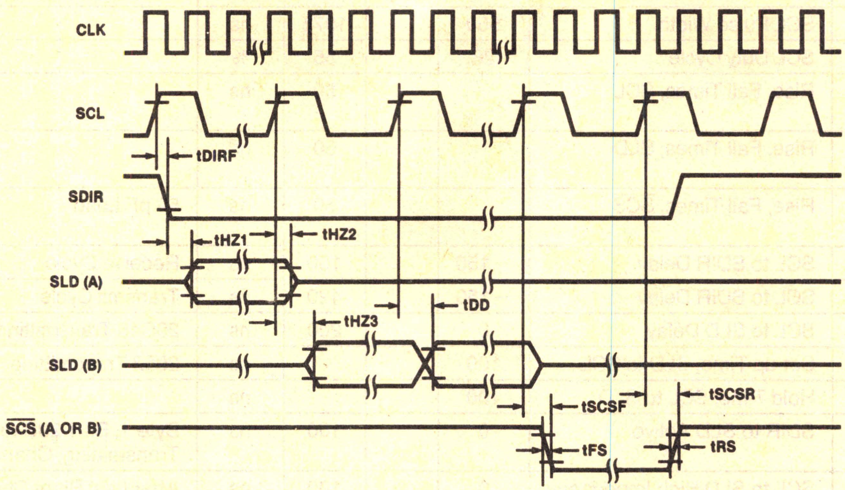
TIMING PARAMETERS (CLK = 1.544 MHz, 33% DUTY CYCLE)

RECEIVE CYCLE



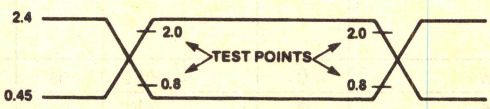
270153-23

TRANSMIT CYCLE



270153-24

A.C. TESTING INPUT, OUTPUT WAVEFORM



270153-25

A.C. Testing inputs are driven at 2.4 for a logic "1" and 0.45 for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

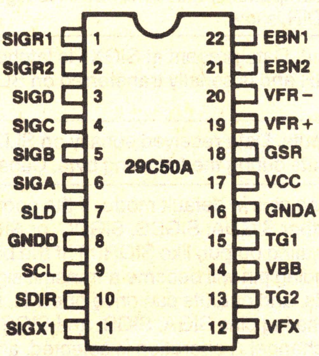
iATC 29C50A
FEATURE CONTROL COMBO

- **External and User Programmable Transmit and Receive Gain**
- **Programmable Internal and External Hybrid Balance Network Select**
- **Programmable Analog, Digital, and Subscriber Loopback**
- **Programmable μ/A-Law Select**
- **Flexible Signaling Interface**
- **Pin Selectable Channel A or B Operation on the SLD Interface**
- **Three Party Conferencing**
- **Low Power Consumption**

The Intel iATC 29C50A Feature Control Combo is an advanced user-programmable, fully integrated PCM Codec with transmit/receive filters fabricated in CHMOS technology. This technology allows the 29C50A to provide excellent transmission performance while achieving low power consumption.

The 29C50A supports the analog subscriber with a variety of added per-line features to the normal BORSCHT functions associated with the analog line circuit. Some of these features include programmable transmit and receive gain, on-chip or custom hybrid balancing network selection and interpolation, a flexible signaling interface, and programmable μ or A-Law conversions. Additionally, the 29C50A can operate on either the A or B channel of the SLD interface, allowing two combos to be connected to one SLD link.

The 29C50A is intended for use with the 2952 Integrated Line Card Controller in digital switching applications. The 2952 handles the transfer of voice, feature control, and signaling information between the backplane and up to 16 29C50A combos. The 29C50A is also suited for use in ISDN terminal applications.



230982–1

Figure 1. Pin Configuration

September 1986
Order Number: 230982-003

Table 1. Pin Description

Symbol	Function
V_{CC}	Most positive supply; input voltage is +5V ±5%.
V_{BB}	Most negative supply; input voltage is −5V ±5%.
GNDA	Analog ground return line. Not internally connected to GNDD.
GNDD	Digital ground return line. Not internally connected to GNDA.
VFX	Analog voice input to transmit channel.
TG1	Inverting input to transmit gain adjusting op-amp. Feedback point for external gain adjusting resistor network up to 10 KΩ.
TG2	Output of the transmit gain adjusting op-amp. Will drive external gain adjusting resistor network up to 10 KΩ.
VFR+	Non-inverting output of the power amplifier. Capable of directly driving transformer hybrids or high impedance loads either single ended or differentially.
VFR−	Inverting output of power amplifier. Capable of directly driving transformer hybrids or high impedance loads either single ended or differentially.
GSR	Input to receive gain setting circuit. An external resistor network connected between VFR− and VFR+, and GSR sets the receive channel gain from 0 dB to −9.54 dB. Connecting GSR to GNDA will set the gain at −6.02 dB.
EBN1	Input for the first external balance network.
EBN2	Input for the second external balance network.
SCL	Subscriber clock. Supplied by the 2952 line card controller, this is a 512 KHz, 50% or 33% duty cycle clock. Input will accept TTL levels.
SDIR	Subscriber direction signal and frame sync. When high, SLD becomes an input and data is transferred from the 2952 to the 29C50A. When low, the output buffer on the 29C50A SLD pin is enabled and data is transferred from the 29C50A to the 2952. Input will accept TTL levels.
SLD	Subscriber data link. A 512 Kbps bi-directional serial data port, which is clocked by SCL. SLD becomes a TTL compatible input when SDIR is high and an output capable of driving one TTL load when SDIR is low.
SIGX1	Transmit signaling input. Data present at SIGX1 is latched by an internal signal preceding the falling edge of SDIR and is serially transferred on SLD during the transmit signaling byte. TTL compatible.
SIGR1 SIRG2	Receive signaling outputs. Data received serially on SLD during the receive signaling byte is latched on these outputs during the following byte. Capable of driving one TTL load.
SIGA SIGB SIGC SIGD	Programmable signaling pins in default mode. If the appropriate bit in the feature control memory is set high (either SIGDA, SIGDB, SIGDC, or SIGDD), the corresponding pin will become a receive signaling output, like SIGR(n). If the bit in the feature control memory is set low, the corresponding pin will become a transmit signaling input, like SIGX1. Inputs will accept TTL level inputs, and outputs can drive one TTL load. During channel A/B operation, the programmable signaling pins SIGA, SIGB, and SIGC take on different functions. If SIGA is connected to V_{BB}, channel A operation is selected, and SIGB functions as an output only. If SIGB is tied to V_{BB}, channel B operation is selected, and SIGA functions as an output only. SIGC becomes a transmit only pin for both channel A and channel B operation. SIGD remains programmable.

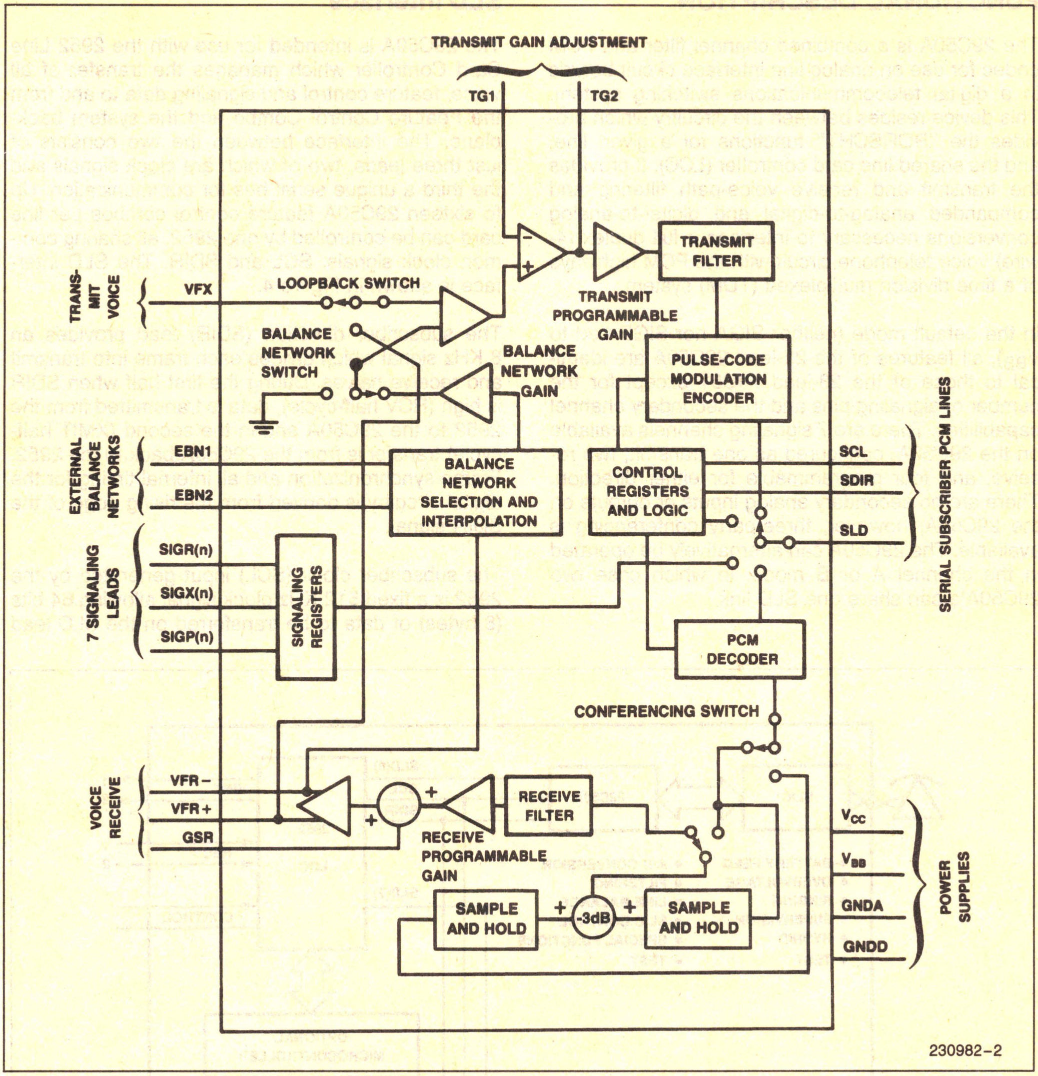


Figure 2. Block Diagram

FUNCTIONAL DESCRIPTION

The 29C50A is a combined channel filter and PCM codec for use on analog line interface circuit boards in a digital telecommunications switching system. This device resides between the circuitry which provides the "BORSCHT" functions for a given line, and the shared line card controller (LCC). It provides the transmit and receive voice-path filtering and companded analog-to-digital and digital-to-analog conversions necessary to interface a full duplex (4-wire) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system.

In the default mode (neither SIGA nor SIGB tied to V_{BB}), all features of the 22-lead 29C50A are identical to those of the 28-lead 29C51 except for the number of signaling pins and the secondary channel capabilities. There are 7 signaling channels available on the 29C50A, configured as one transmit, two receive, and four programmable for either direction. There are no secondary analog inputs or outputs on the 29C50A; however, three-party conferencing is available. The 29C50A can alternatively be operated in the channel A or B mode, in which case two 29C50A's can share one SLD link.

SLD Interface

The 29C50A is intended for use with the 2952 Line Card Controller which manages the transfer of all voice, feature control and signaling data to and from the Feature Control Combo and the system backplane. The interface between the two consists of just three leads, two of which are clock signals and the third a unique serial bus for communication. Up to sixteen 29C50A feature control combos per line card can be controlled by one 2952, all sharing common clock signals, SCL and SDIR. The SLD interface is shown in Figure 4.

The subscriber direction (SDIR) lead provides an 8 KHz signal which divides each frame into transmit and receive halves. During the first half when SDIR is high (RCV half-cycle), data is transmitted from the 2952 to the 29C50A and in the second (XMIT half-cycle) transfer is from the 29C50A back to the 2952. Frame synchronization and all internal timing for the digital circuitry is derived from the rising edge of the SDIR signal.

The subscriber clock (SCL) input generated by the 2952 is a fixed 512 KHz clock signal allowing 64 bits (8 bytes) of data to be transferred on the SLD lead

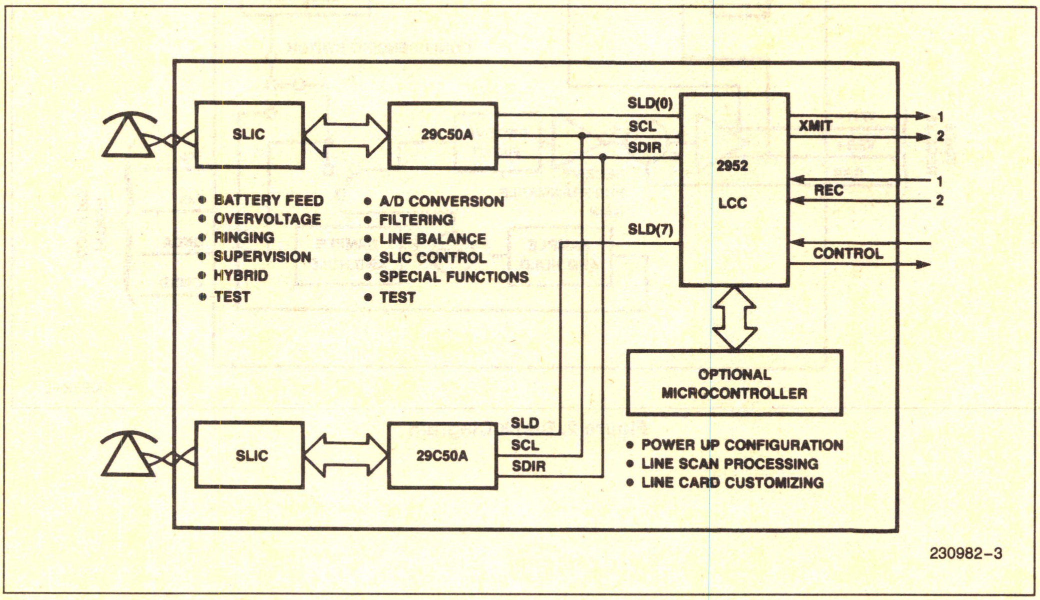


Figure 3. Analog Linecard

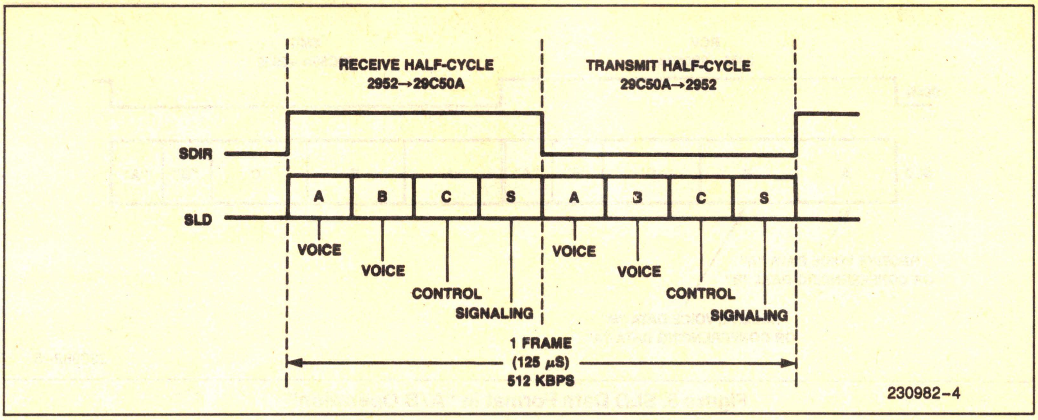


Figure 4. 29C50A/2952 Interface

during each 125 μs frame. Depending on 2952 master clock frequency, the SCL duty cycle can be either 50% or 33%.

The subscriber data link (SLD) is a bi-directional serial bus that transfers eight bytes of serial data to and from the 29C50A each frame. During the first half of each frame, RCV channel information is transferred to the 29C50A in four bytes consisting of channel A voice, channel B voice, feature control, and signaling information. Similarly during the second half-cycle, four bytes of XMIT channel information are sent to the 2952. The MSB (bit 7) of each byte is sent first on the SLD. After the last valid signaling bit is transmitted to the 2952, the bus is placed in a high impedance state for at least one SCL clock cycle to prevent data contention on the bus. (See FCB #6—Signaling Register.)

Upon power supply application and clocks SCL and SDIR applied, the 29C50A will automatically enter the power down state. During the transmit half-cycle (29C50A talking to the 2952) a code of all ones will be sent to the controller during the channel A or B byte depending upon the selected mode. The 29C50A SLD pin will be placed in a high impedance state during the unused channel.

Channel A/B Operation

The 29C50A can use either channel A or channel B on the SLD interface for transfer of PCM voice. This allows two 29C50A combos to share a single SLD link, allowing up to sixteen combos to be controlled by one 2952 line card controller. The channel A/B mode is selected with the SIGA and SIGB pins. If SIGA is connected to V_{BB}, the 29C50A will use channel B. If neither pin is connected to VBB, the 29C50A will operate identically to the 29C51 Feature Control Combo, except for the lack of secondary analog channels. Connecting both SIGA and SIGB to V_{BB} is not allowed. Operation of the combo on the SLD interface is shown in Figure 5.

When programming the 29C50A, the least significant bit (last bit transmitted) of the first control byte of a programming frame selects the 29C50A which will accept this and all following bytes of the frame. Only the LSB of this first framing byte is used as a channel select bit. It is latched until the next framing byte (00 or 01 header) is received in the feature control channel. If this bit is a zero, the channel A combo will accept the programming data, and if this bit is a one, the channel B combo will accept the data. Only the selected combo will echo back the programming data.

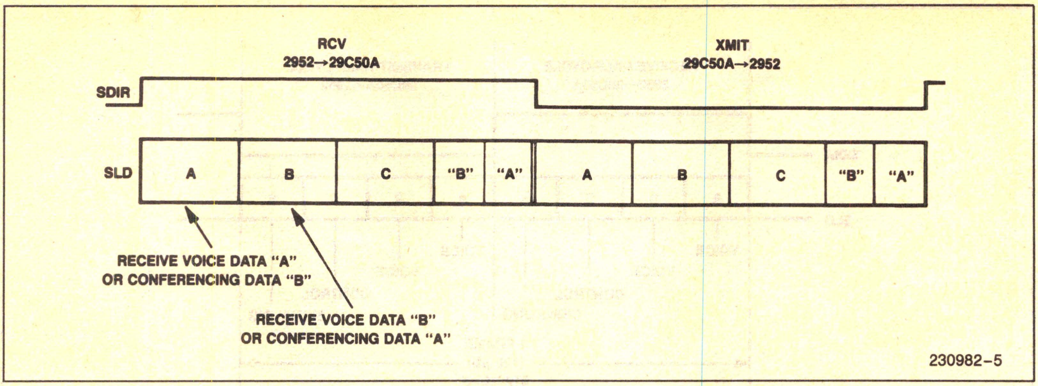


Figure 5. SLD Data Format in "A/B Operation"

Similarly, when verifying the feature control memory of the 29C50A, the desired device is selected using the least significant bit of the first byte of the verification frame. Only the selected device will respond. Because the LSB of the feature control framing byte must be set to a one for verification of a channel B device, the 2952 will actually read back 10 bytes from the SLD interface when using the bi-directional FIFO (BFF). Only six of these bytes will contain the 29C50A programming information. Refer to the 2952 Reference Manual for further details regarding verification of feature control information from an SLD slave device using the 2952.

When neither device is being programmed or verified, the last selected device will echo the received control channel data during the transmit half of the SLD frame.

In order to take full advantage of the last look change detection logic of the 2952, the two 29C50A combos using one SLD line share the signaling channels each frame. Each uses a nibble of the transmit and receive signaling bytes. Since one signaling pin is used to select channel A or B operation, six pins are available for carrying signaling data, with three configured as receive, two as transmit, and one programmable.

For a channel A device, data received during bits 3, 2, 1, and 0 of byte 4 on the SLD will appear at SIGR1, SIGR2, SIGB, and SIGD (if SIGD is chosen as a receive bit), respectively. For a channel B device, the data from bits 7, 6, 5, and 4 of byte 4 appears at SIGR1, SIGR2, SIGA, and SIGD (if SIGD is chosen as a receive bit), respectively.

In the transmit direction, a channel A part will send data from SIGX1, SIGC, and SIGD (if SIGD is chosen as a transmit bit) in bit positions 3, 2, and 1 respectively of byte 8 on the SLD. A channel B part will send data from those same pins in bit positions 7, 6, and 5 respectively of byte 8. Neither combo will drive the SLD line during the third or seventh bits.

Figure 4 shows operation of the 29C50A on the SLD interface in the default mode, when neither the channel A nor channel B mode is selected (neither SIGA nor SIGB connected to V_{BB}).

Signaling field formats for all three modes are shown in Figure 11.

TRANSMIT AND RECEIVE OPERATION

Transmit Filter

A low pass anti-aliasing section is included on chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. The 29C50A specifications meet the digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 12.

A high pass section configuration rejects low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Gain of up to 20 dB can be set without degrading the performance of the filter.

Encoding

The output of the transmit filter is internally sampled by the encoder and held on an internal sample and hold capacitor. DC offset is corrected by an on-chip auto zero circuit. The signal is then encoded and presented as PCM data on the SLD lead on the first 8 bits of the XMIT half frame (fifth byte) for channel A operation, or the second 8 bits (sixth byte) for channel B operation.

Decoding

The PCM words received on the SLD are demultiplexed and sent to the decoder. For channel A operation, the first SLD byte is decoded. For channel B operation the second SLD byte is used. The decoded value is held on an internal sample and hold capacitor. If, however, conferencing has been selected, both the first and second SLD bytes will be decoded, added, and subsequently passed to the receive filter.

Receive Filter

The receive section of the filter provides a passband flatness and stopband rejection which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. The receive filter transfer characteristics and specifications will be within the limits shown in Figure 13.

GENERAL OPERATION

External Gain Setting

Both transmit and receive gain levels are factory trimmed, but can be modified by external resistors during line card assembly. The value of transmit gain is adjusted by connecting resistors RT1 and RT2 (see Figure 6) at the two external gain setting control pins, TG1 and TG2. These two pins are the input and output of an on-board gain amplifier stage, and the resistors provide the necessary input and feedback for gain control. The value of external gain is given by:

$$A = 1 + RT1/RT2$$

For unity gain, pins TG1 and TG2 are tied together. Similarly, for the receive section, external resistors RR1 and RR2 at pins VFR+, GSR, and VFR− set the external gain given by:

$$A = (RR1 + RR2)/(RR1 + 3RR2)$$

A value greater than 10 KΩ and less than 100 KΩ for R1 + R2 is recommended. The output is capable of driving loads of 300Ω at 3.2Vp single ended or 600Ω at 6.4Vp differentially.

Three additional gain settings of 0 dB, −6 dB, and −9.54 dB can be realized without using any external components by strapping pin GSR to VFR−, GNDA, and VFR+, respectively.

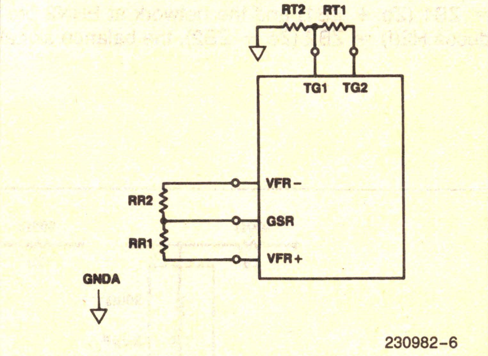


Figure 6. External Gain Connections

Hybrid Balancing Network

The 2- to 4-wire conversion necessary for subscriber interface is partially integrated on-chip. Network line balancing needed to minimize the trans-hybrid loss from the receive to transmit direction analog signals is handled internally. The three internal networks shown in Figure 8 may be selected by programming the appropriate feature control byte. These networks are integrated in a switched capacitor configuration and have single pole-zero characteristics in the 200 Hz to 3200 Hz range. They were chosen to serve a wide base of U.S. an European requirements, and can be used as standard line balancing networks or as test networks.

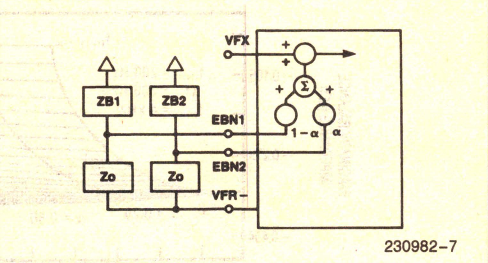


Figure 7. External Balance Network and Interpolation Configuration

Additionally, the user may apply two external balance networks to accommodate varying subscriber loop characteristics (see Figure 7 for external connections). Presumably, these two networks can represent the two extremes of line conditions in different applications such as long or short loops and loaded or unloaded lines. To serve typical lines with characteristics in between the two extremes, the interpolation capability provides a weighted average of the network frequency characteristics. If the external network at EBN1 produces a transfer function $H1(f) = ZB1\ (Zo + ZB1)$ and the network at EBN2 produces $H2(f) = ZB2\ (Zo + ZB2)$, the balance signal can be programmed to have the transfer function H(f):

$$H(f) = \alpha\ H1(f) + (1 - \alpha)\ H2(f)$$

where "α" is the interpolation coefficient programmed to have any of the five values of 0, 0.25, 0.50, 0.75, or 1.0. Figure 7 displays how the subtraction of the coupling signal is implemented inside the device.

As an example, the two external networks shown in Figure 9 represent typical hybrid balance networks for loaded (ZB1) and unloaded (ZB2) analog loops. The graph in Figure 9 shows the real and imaginary components of the equivalent impedance of these two networks as a function of frequency and the interpolation coefficient.

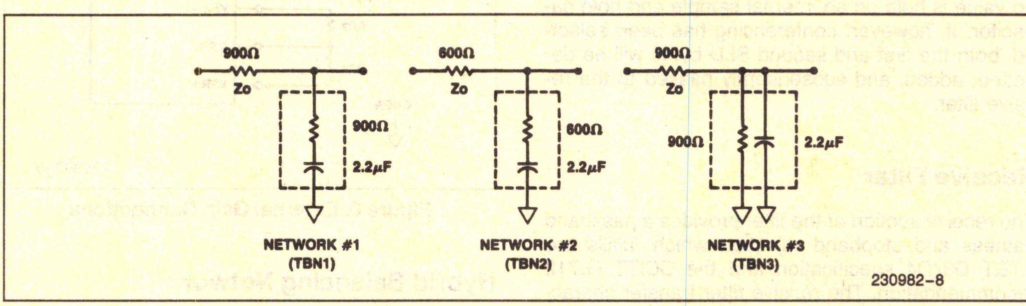


Figure 8. Internal Balance Networks

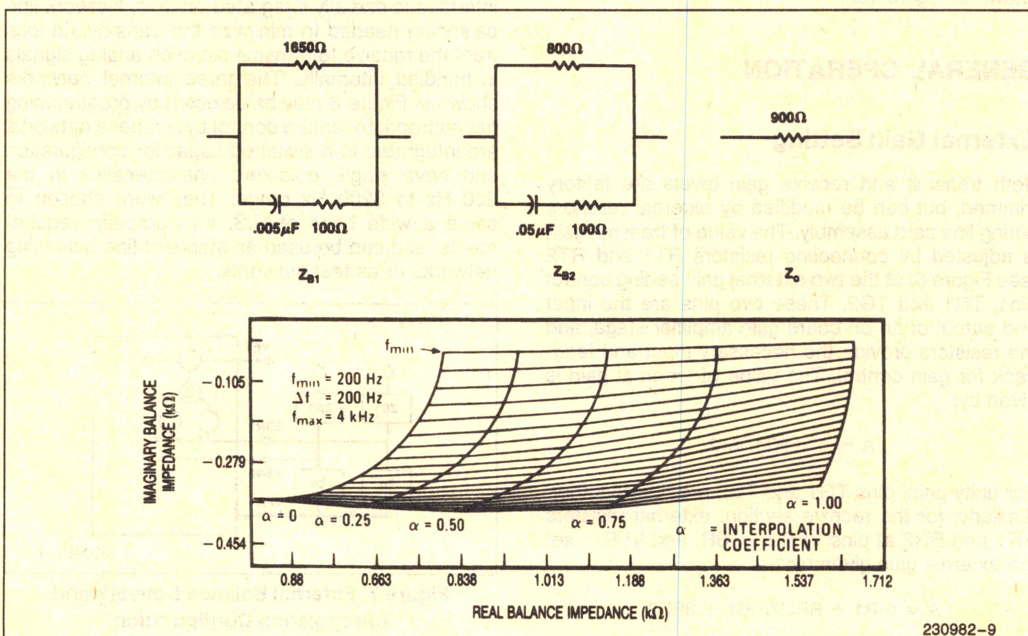


Figure 9. Typical External Balance Networks and Complex Impedance Plot

Conferencing

The 29C50A supports three party conferencing in the default and channel A and B modes. In all three modes, the PCM words received in the A and B channels on the SLD are converted to analog values and added together before being applied to the receive PCM filter. The sum will be attenuated by 3 dB.

In the default mode, the 29C50A will repeat the channel A transmit direction PCM voice byte in channel B.

Precision Voltage References

Voltage references are generated on-chip and are trimmed during the manufacturing process. Separate references are supplied for both the transmit and receive sections of the chip, each trimmed independently. These references determine the gain and dynamic range of the device and provide the user a significant margin for error in other board components.

PROGRAMMABLE FEATURES

The 29C50A is configured by the 2952 line card controller by a set of six feature control bytes (FCB). These bytes of information are stored in internal registers which are serially multiplexed to and from the SLD interface in the third and seventh byte locations. The first two bits of each byte consist of a multiframe synchronization and write enable code. The framing bit (bit 7, MSB) establishes the beginning of a feature control frame when set to a logical zero, and increments the feature control counter when set to one. The second (bit 6) enables the writing to the 29C50A when it is the logical complement of the framing bit.

When writing new feature control information to the 29C50A, the first byte should contain a framing (F) and write enable (WE) header of 01 (F=0 and WE=1). This designates a new frame of information to transfer. The subsequent bytes should each have F = 1 to advance the counter, and WE = 0 to enable the write operation.

The controller can also request to verify the feature control register contents by sending a 00 or 11 at the beginning of the byte to be read. To read the first byte, a 00 F/WE code should be sent while each subsequent byte should have a 11 header. An internal six-stage counter is set on the first byte verified then incremented once each 125 μs frame. It is reset only upon detection of a 01 or 00 F/WE. Once the counter is greater than six, neither read nor write modes may be selected by sending the 29C50A a 11 framing and write enable code. The 29C50A will then echo in byte 7 the data it received in byte 3.

The LSB of the first feature control byte selects whether the channel A or channel B device will accept the feature control commands. If this bit is 0, the channel A device is selected. If this bit is 1, the channel B device is selected. Only the selected device will echo feature control data.

FCB #1—Power Up/Down, Loop Back Mode, μ/A-Law Select Register

POWER UP AND DOWN

The 29C50A can be instructed to go into the power down or standby mode for reduced power consumption. In this mode, all analog inputs and outputs are placed in a high impedance state, inhibiting all voice signals. A code of all ones will be output in the voice byte on the SLD. Signaling and feature control information will continue to be processed to allow the 29C50A to be read or reprogramed, and to allow the backplane to monitor the subscriber line.

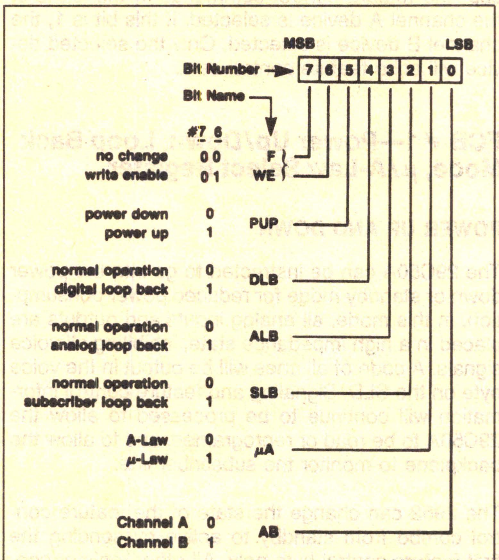
The 2952 can change the state of the feature control combo from standby to active by sending the first feature control byte only. All other register contents will be preserved during power down provided the power supplies remain connected.

LOOP BACK MODE SELECT

Three modes of remote testing are incorporated in the 29C50A and can be selected by appropriate coding in this register. The loopback features allow a number of tests to be performed to determine line quality and balancing. These include digital loop back, analog loop back, and subscriber loop back.

In the digital loopback mode, the combo retransmits the PCM words it receives in the voice byte of the SLD back to the line card controller in the same frame. This feature allows path verification and testing of the circuit up to the slave device.

When the analog loopback mode is selected, the analog output VFR+ is internally connected to the analog input VFX. This feature allows functional testing of the combo as well as gain adjustment.



230982-10

In the third test mode, subscriber loopback, the digital output of the A/D converter is internally connected to the input of the D/A converter. The analog signal input to VFX is sent through the transmit filter, encoded, then decoded, filtered and output to VFR+ and VFR−. This mode is used primarily for simplifying analog to analog testing from the subscriber side of the line card.

CONVERSION LAWS

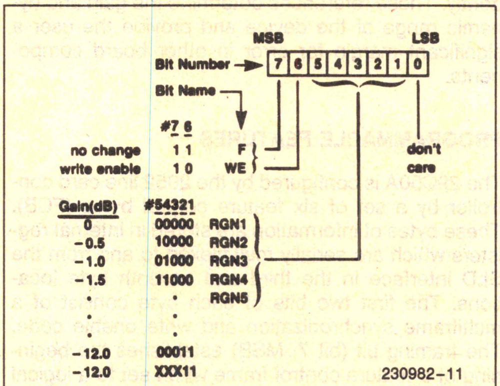
The 29C50A can be selected for either μ-law or A-law operations. A user can select either conversion law by assigning the corresponding bit. A logical 1 in bit 1 would select μ-law while a logical 0 would select A-law conversions. Both conversions follow CCITT recommendation G.711.

CHANNEL SELECTION

The LSB of the first feature control byte determines whether the channel A or channel B device will accept the feature control commands. This bit has no effect if the channel A/B operation is not selected.

FCB #2—Receive Programmable Gain Register

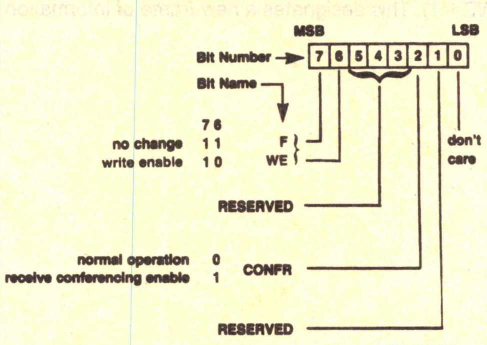
The receive gain levels can be adjusted by applying external resistors as mentioned earlier, or by selective programming of this register. A range from 0 to −12 dB in 0.5 dB increments can be realized for the receive channel.



230982-11

FCB #3—Conferencing Register

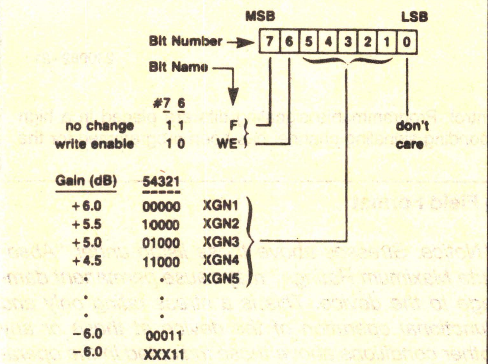
The 29C50A can be programmed to decode both receive voice channels and add the two analog signals to perform three-party conferencing.



230982-12

FCB #4—Transmit Programmable Gain Register

The gain setting of the transmit section of the chip operates in the same manner as the receive gain register. A 12 dB range from −6.0 dB to +6.0 dB in 0.5 dB increments is available.



230982-13

FCB #5—Balance Network Select and Gain Register

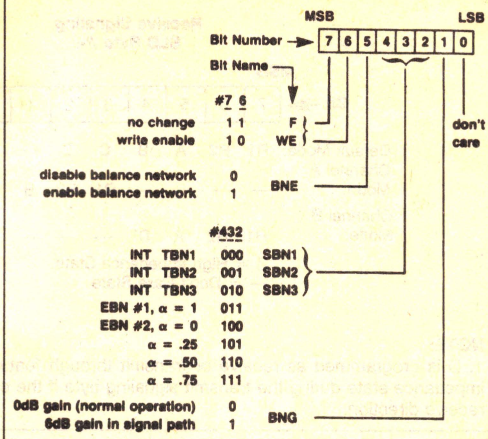
BALANCE NETWORKS

The 29C50A offers a choice of internal or external hybrid balancing. Externally, two balance networks connected to pins EBN1 and EBN2 can be used independently, or as a weighted average of the two. The weighting factor, or interpolation coefficient, can range from 0 to 1 in steps of 0.25. Setting "α" to be 1 or 0 results in selecting either EBN1 or EBN2 respectively.

Three additional balance network configurations consisting of either a series or parallel RC circuit are located internal to the device. (See Figure 8.)

GAIN SETTING

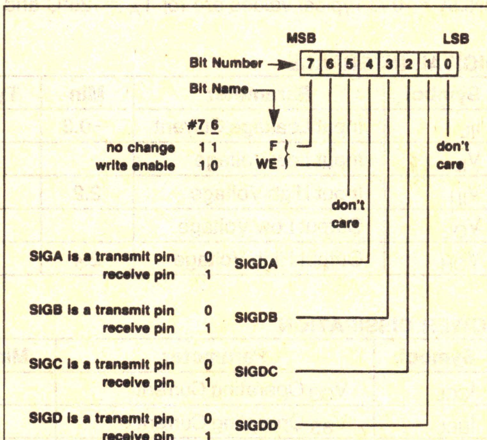
An additional 6 dB gain in the balance signal path can be realized by coding this bit with a logical one. A logical zero provides unity gain.



230982-14

FCB #6—Signaling Register

Four pins are provided on the 29C50A to be used as selectable transmit or receive signaling inputs. A code of one in the respective bit commits the pin to receive signal information and a zero to transmit. The signaling field format as it appears on the SLD bus is shown in Figure 11 for both channel A/B and default operation. R1 and R2 correspond to signaling information received on SIGR1 and SIGR2, respectively. Similarly, programmable pins SIGA, SIGB, SIGC, SIGD, and transmit pin SIGX1 are coded into the bit location as shown below. Bits 2, 3 and 4 have no effect in channel A/B operation.



230982-15

Receive Signaling SLD Byte #4 (MSB … LSB)

Bit →	7	6	5	4	3	2	1	0
Default Mode:	R1	R2	A	B	C	D	—	—
Channel A Mode:	—	—	—	—	R1	R2	B	D[1]
Channel B Mode:	R1	R2	A	D[1]	—	—	—	—

Transmit Signaling SLD Byte #8 (MSB … LSB)

Bit →	7	6	5	4	3	2	1	0
Default Mode:	X1	A	B	C	D	Z	Z	Z
Channel A Mode:	Z	Z	Z	Z	X1	C	D[1]	Z
Channel B Mode:	X1	C	D[1]	Z	Z	Z	Z	Z

Z = High Impedance State
— = Don't Care State

230982–21

NOTE:
1. D is programmed as receive or transmit through feature control. Programmable signaling bits are placed in a high impedance state during the transmit signaling byte if the corresponding signaling channel has been programmed for the receive direction.

Figure 11. Signaling Field Format

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
All Input and Output Voltages with Respect to V_{BB} .... −0.3V to 13V
All Input and Output Voltages with Respect to V_{CC} .... −13V to 0.3V
Power Dissipation .... 1.35 W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

DC CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%; SCL (50% duty), SDIR, SLD applied GNDD = 0V, GNDA = 0V. Typical values are for T_A = 25°C and nominal power supply values

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current	−0.3		±10	μA	$V_{BB} \leq V_{IN} \leq V_{CC}$
V_{IL}	Input Low Voltage			0.8	V	
V_{IH}	Input High Voltage	2.2		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	$I_{OL} \geq -1.6$ mA, 1 TTL Load
V_{OH}	Output High Voltage	2.4			V	$I_{OH} \leq 50$ μA, 1 TTL Load

POWER DISSIPATION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{CCL}	V_{CC} Operating Current		9		mA	
I_{BBL}	V_{BB} Operating Current		9		mA	
I_{CCO}	V_{CC} Standby Current		0.8		mA	
I_{BBO}	V_{BB} Standby Current		0.8		mA	
P_{D0}	Standby Power Dissipation		8		mW	
P_{DI}	Operating Power Dissipation		90		mW	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

TG1 = TG2, Transmit Programmable Gain = 6 dB. Receive Programmable Gain = 0 dB

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response		±0.1		dB	Signal Input of 0 dBm0 f = 1.02 KHz
DmW	Digital Milliwatt Response		±0.1		dB	f = 1.02 KHz
$DmW_{\mu V}$	Digital Milliwatt Response VFR+, VFR− μ-law		6.14 1.571		dBm Vrms	VFR + Single-Ended Output R_L = 600Ω Receive Input per CCITT G.711
DmW_{AV}	Digital Milliwatt Response VFR+, VFR− A-law		6.17 1.576		dBm Vrms	
$OTLP_{\mu X}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.116 0.785		dBm Vrms	μ-law, Referenced to 600Ω
$OTLP_{AX}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.149 0.788		dBm Vrms	A-law, Referenced to 600Ω

GAIN TRACKING

Reference level = 0 dBm0 for μ-law, −10 dBm0 A-law at 1.02 kHz, TG1 = TG2, GSR = VFR−, Transmit Programmable Gain = 6 dB, Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
GT_T	Transmit Gain Tracking Error Sinusoidal Input; μ or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
GT_R	Receive Gain Tracking Error Sinusoidal Input; μ or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 AT&T PUB43801 and CCITT G.712—Method 2

ANALOG INTERFACE, RECEIVE CHANNEL

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
R_{OR}	Output Resistance, VFR+ VFR−		1		Ω	
V_{OSR1}	Output Offset, VFR+ or VFR−, Single Ended		50		mV	Relative to GNDA
V_{OSR2}	Output Offset, VFR+ to VFR−, Differential		75		mV	
C_{LR}	Load Capacitance, VFR+, VFR−,			100	pF	
V_{OR1}	Max Output Voltage Swing Across R_L, VFR+, VFR−, Single-Ended Connection	±3.2			Vp	R_L ≥ 300Ω
V_{OR2}	Max Differential Output Voltage Swing, VFR+, VFR−	±6.4			Vp	R_L ≥ 600Ω
P_{OR}	Differential Output Power, VFR+, VFR−			15.3	dBm	R_L = 600Ω

ANALOG INTERFACE, TRANSMIT CHANNEL

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{BX}	Input Leakage Current, EBN1, EBN2, TG1		100		nA	$-1.6V < VFX < 1.6V$
R_{IX1}	Input Resistance, VFX		500		KΩ	$-1.6V < VFX < 1.6V$
R_{IX2}	Input Resistance, EBN1, EBN2, TG1		10		MΩ	$-1.6V < VFX < 1.6V$
TGmax	Max Transmit Gain Adjust			20	dB	
V_{OTG}	Max Output Voltage Swing TG2			±3.2	V	$R_L \geq 10K\ \Omega$, (Note 1)
C_{LX}	Load Capacitance, TG2			20	pF	
R_{LX}	Load Resistance, TG2	10			KΩ	

NOTE:
1. Transmit programmable gain must be set at −6 dB to encode this level without clipping.

DISTORTION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
SD_X, SD_R	Signal to Distortion, μ or A-law Sinusoidal Input; CCITT G.712— Method 2 Half Channel	35 29 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X, DP_R	Single Frequency Distortion Products In Band (2nd or 3rd Harmonic Half Channel)		−50	−46	dB	Input = 1.02 KHz 0 dBm0 AT&T Advisory #64 (3.8)
IMD_1	Intermodulation Distortion, End to End Measurement			−40	dBm0	CCITT G.712(7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−50	dBm0	CCITT G.712(7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−27	dBm0	CCITT G.712(6.1)
SIS	Spurious In Band Signals, End to End Measurement			−40	dBm0	CCITT G.712(9)
D_{AX}	Transmit Absolute Delay		180		μs	0 dBm0, 1.02 KHz Includes Delay Through A/D
D_{DX}	Transmit Differential Envelope Delay; Relative to Minimum Envelope Delay (1.4 KHz)		170 95 45 105		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz
D_{AR}	Receive Absolute Delay		125		μs	0 dBm0, 1.02 KHz Includes Delay Through D/A
D_{DR}	Receive Differential Envelope Delay; Relative to Minimum Envelope Delay (300 Hz)		45 35 85 110		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz

NOISE (Primary Channel)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted		12		dBrnCO	Transmit Gain Adjust = 0 dB
N_{XP1}	Transmit Noise, Psophometrically Weighted		−78		dBm0p	Transmit Gain Adjust = 0 dB
N_{RC1}	Receive Noise, C-Message Weighted		10		dBrnCO	Unity Gain; Idle Code
N_{RPI}	Receive Noise, Psophometrically Weighted		−80		dBm0p	Unity Gain; Idle Code
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit Channel		−35		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).
$PSRR_2$	V_{BB} Power Supply Rejection Transmit Channel		−30		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).
$PSRR_3$	V_{CC} Power Supply Rejection, Receive Channel		−35		dB	Idle Channel, 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).
$PSRR_4$	V_{BB} Power Supply Rejection Receive Channel		−30		dB	Idle Channel, 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).

CROSSTALK

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
CT_{TR}	Crosstalk, Transmit Voice to Receive Voice		−75		dB	Input = 0 dBm0, Unity Gain 1.02 KHz; Idle Code on SLD Voice Byte
CT_{RT}	Crosstalk, Receive Voice to Transmit Voice		−75		dB	0 dBm0, 1.02 KHz Signal at SLD Receive Voice Byte; VFX = GNDA

NOTE:
1. Measured at SLD Voice bytes for transmit channel. Measured at VFR for receive channel.

TRANSMIT VOICE FREQUENCY CHARACTERISTICS

TG1 = TG2, Transmit Programmable Gain = 6 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at VFX
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−22	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.70		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	
ΔG_{PX}	Programmable Gain Accuracy (Cummulative Error)		±0.25		dB	Freq. = 1.02 KHz for all Steps

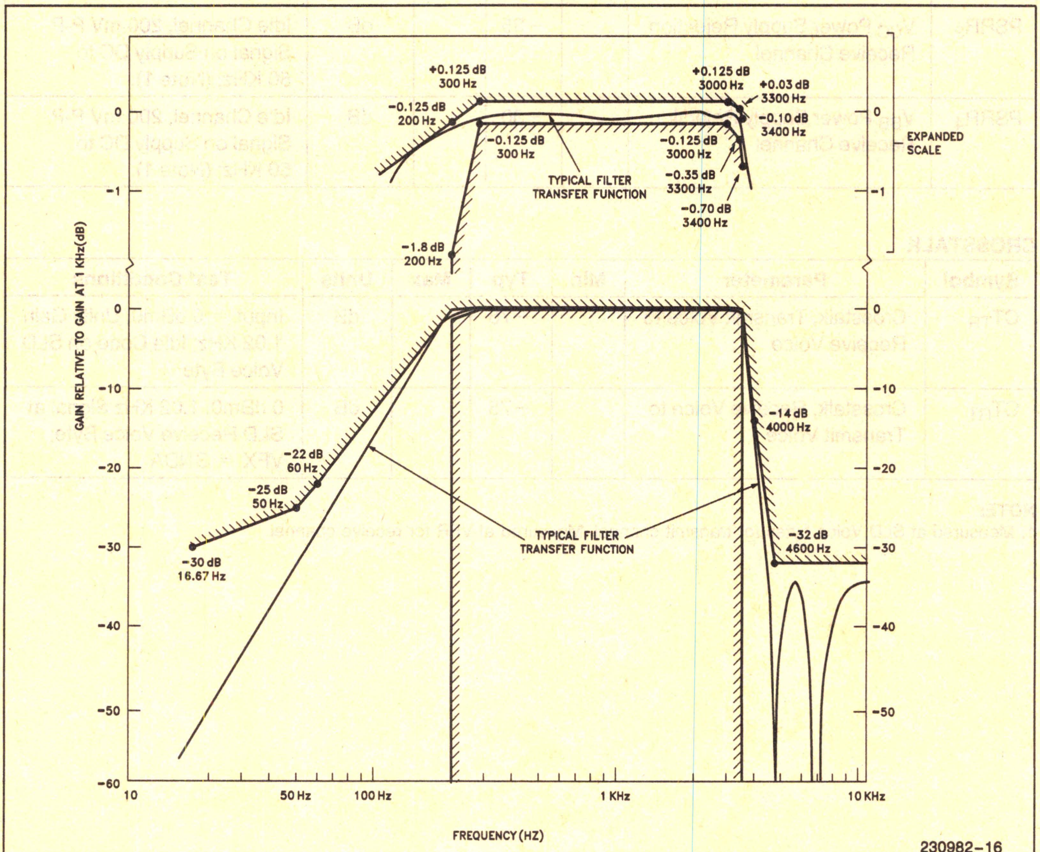
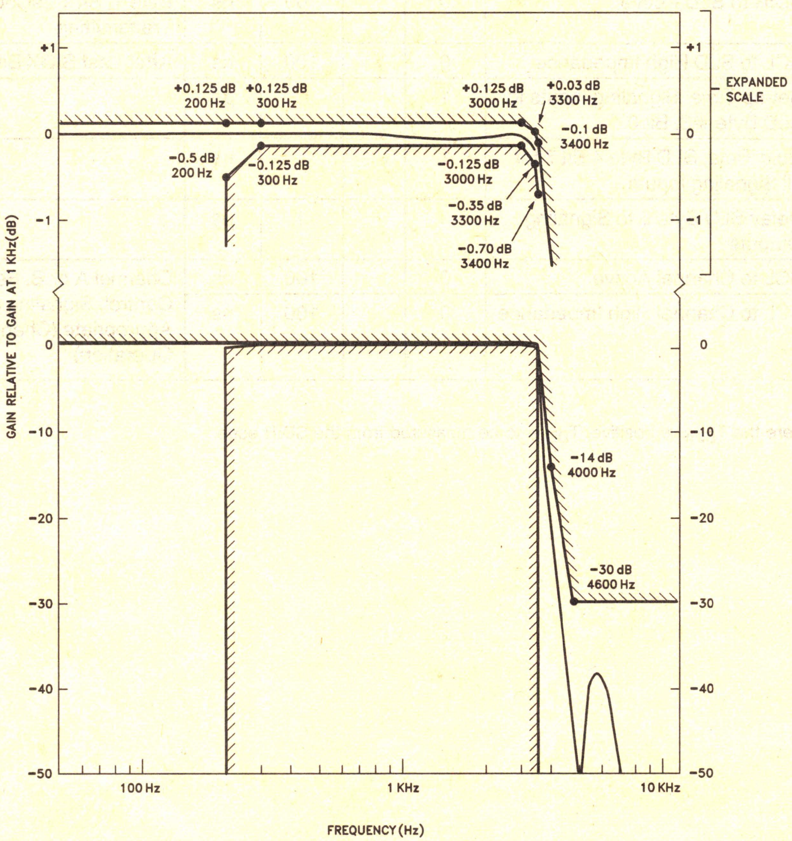


Figure 12. Transmit Voice Frequency Characteristics

RECEIVE VOICE FREQUENCY CHARACTERISTICS

GSR = VFR−, Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Input on SLD
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.70		−0.1	dB	
	4000 Hz 4600 Hz & Above			−14 −30	dB dB	
ΔG_{PR}	Programmable Gain Accuracy (Cummulative Error)		+0.25	dB	f = 1.02 KHz All Steps	



230982-17

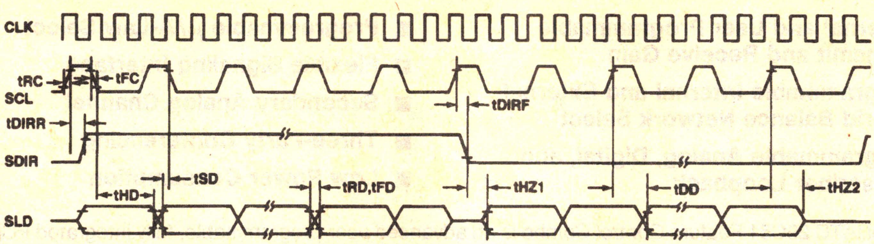
Figure 13. Receive Voice Frequency Characteristics

A.C. CHARACTERISTICS—TIMING PARAMETERS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
T_{DC}	SCL Duty Cycle	28	33	38	%	2952 CLK Clock = 1.544 or 1.536 MHz
		45	50	55	%	2952 CLK Clock ≥ 2.048 MHz
T_{RC} T_{FC}	Rise, Fall Times, SCL			50	ns	
T_{RD} T_{FD}	Rise, Fall Times, SLD			50	ns	
T_{DIRR}	SCL to SDIR Delay	−150		150	ns	
T_{DIRF}	SCL to SDIR Delay	−150		+420	ns	
T_{DD}	SCL to SLD Delay	0		200	ns	29C50A Transmitting (Note 1)
T_{SD}	Set-up Time, SLD to SCL	100			ns	2952 Transmitting
T_{HD}	Hold Time, SCL to SLD	100			ns	
T_{HZ1}	SDIR to SLD Active	0		100	ns	Byte 1, Bit 1 29C50A Transmitting
T_{HZ2}	SCL to SLD High Impedance	0		100	ns	After Last SIGX Bit
T_{SS}	Set-up Time, Signaling Inputs to SLD Byte #3, Bit 0	1			μs	
T_{HS}	Hold Time, SLD Byte 4 Bit 7 for all Signaling Inputs	1			μs	
T_{DS}	Delay SLD Byte 5 to Signaling Outputs			1	μs	
	SCL to Channel Active	0		100	ns	Channel A or B, Feature Control, Signaling as Appropriate (Channel A/B Operation)
	SCL to Channel High Impedance	0		100	ns	

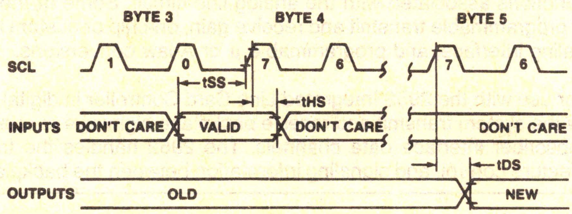
NOTE:
1. In cases where the T_{DIRF} is positive, T_{DD} is to be measured from the SDIR edge.

TIMING PARAMETERS



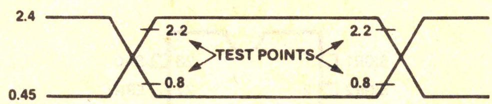
230982-18

SIGNALING TIMING



230982-19

A.C. TESTING INPUT, OUTPUT WAVEFORM



230982-20

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

PRELIMINARY

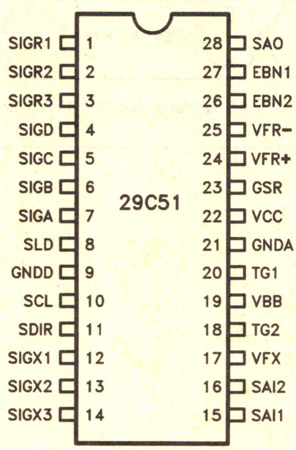
iATC 29C51 FEATURE CONTROL COMBO

- **External and User Programmable Transmit and Receive Gain**
- **Programmable Internal and External Hybrid Balance Network Select**
- **Programmable Analog, Digital, and Subscriber Loopback**
- **Programmable μ/A-Law Select**
- **Flexible Signaling Interface**
- **Secondary Analog Channel**
- **Three-Party Conferencing**
- **Low Power Consumption**

The Intel iATC 29C51 Feature Control Combo is an advanced user-programmable, fully integrated PCM Codec with transmit/receive filters fabricated in CHMOS technology. This technology allows the 29C51 to realize the same excellent transmission performance as in the Intel 2913/2914 combo while achieving the low power consumption typical of CMOS circuits.

The 29C51 Feature Control Combo supports the analog subscriber with a variety of added per-line features to the normal BORSCHT functions associated with the analog line circuit. Some of these features include secondary analog channels, programmable transmit and receive gain, on-chip or custom hybrid balancing network selection, a flexible signaling interface, and programmable μ or A-law conversions.

The 29C51 is intended for use with the 2952 Integrated Line Card Controller in digital switching environments. These components allow the system transmit and receive backplane highways to operate at different frequencies from that of the subscriber interface data channels. The 2952 handles the transfer of primary voice, secondary analog data, feature control, and signaling information between the backplane and up to 8 29C51's.



270239-1

Figure 1. Pin Configuration

November 1986
Order Number: 270239-001

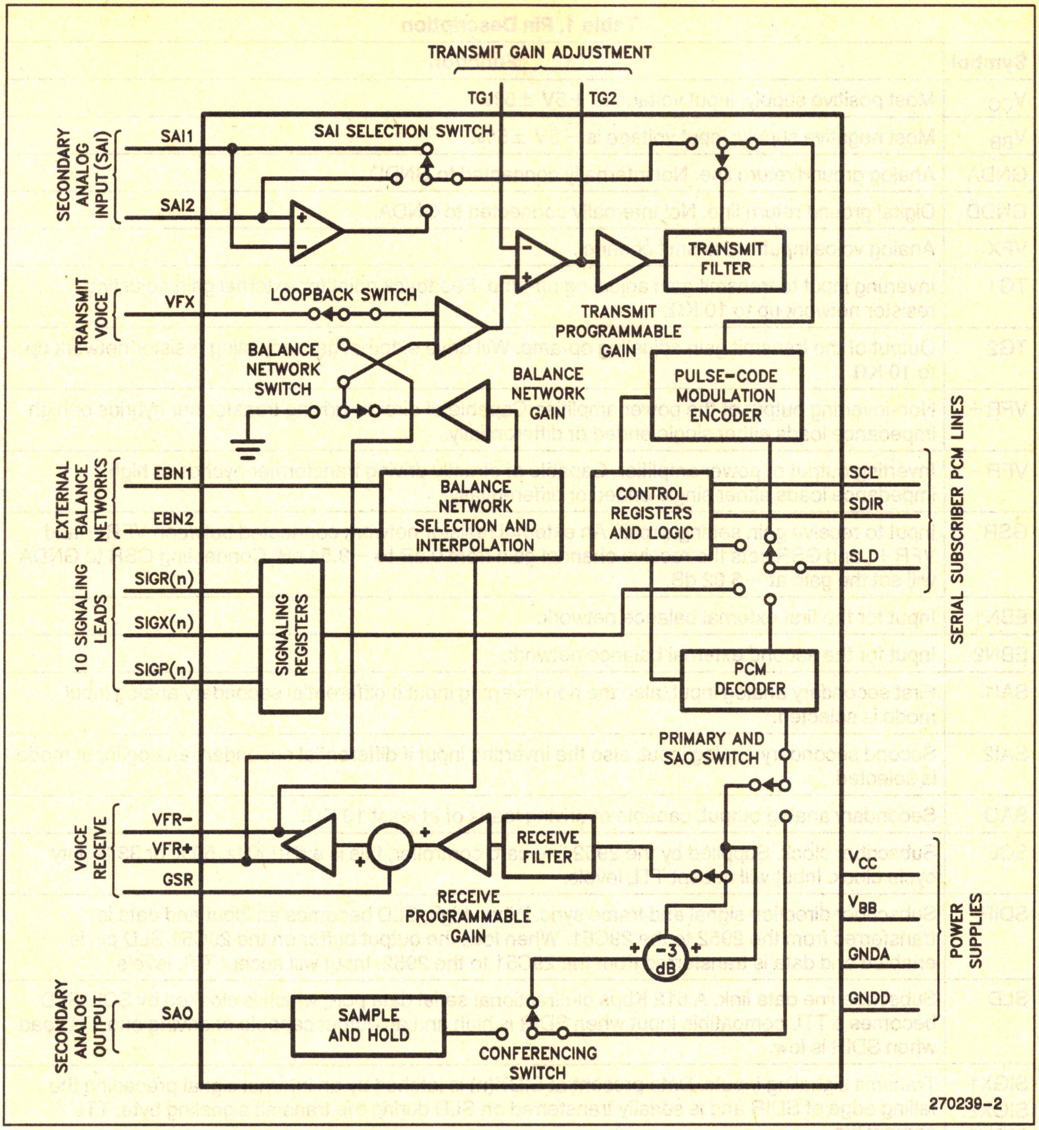


Figure 2. Block Diagram

Table 1. Pin Description

Symbol	Function
V_{CC}	Most positive supply; input voltage is +5V ±5%.
V_{BB}	Most negative supply; input voltage is −5V ±5%.
GNDA	Analog ground return line. Not internally connected to GNDD.
GNDD	Digital ground return line. Not internally connected to GNDA.
VFX	Analog voice input to transmit channel.
TG1	Inverting input to transmit gain adjusting op-amp. Feedback point for external gain adjusting resistor network up to 10 KΩ.
TG2	Output of the transmit gain adjusting op-amp. Will drive external gain adjusting resistor network up to 10 KΩ.
VFR+	Non-inverting output of the power amplifier. Capable of directly driving transformer hybrids or high impedance loads either single ended or differentially.
VFR−	Inverting output of power amplifier. Capable of directly driving transformer hybrids or high impedance loads either single ended or differentially.
GSR	Input to receive gain setting circuit. An external resistor network connected between VFR− and VFR+, and GSR sets the receive channel gain from 0 dB to −9.54 dB. Connecting GSR to GNDA will set the gain at −6.02 dB.
EBN1	Input for the first external balance network.
EBN2	Input for the second external balance network.
SAI1	First secondary analog input, also the non-inverting input if differential secondary analog input mode is selected.
SAI2	Second secondary analog input, also the inverting input if differential secondary analog input mode is selected.
SAO	Secondary analog output, capable of driving loads of at least 10 KΩ.
SCL	Subscriber clock. Supplied by the 2952 line card controller, this is a 512 KHz, 50% or 33% duty cycle clock. Input will accept TTL levels.
SDIR	Subscriber direction signal and frame sync. When high, SLD becomes an input and data is transferred from the 2952 to the 29C51. When low, the output buffer on the 29C51 SLD pin is enabled and data is transferred from the 29C51 to the 2952. Input will accept TTL levels.
SLD	Subscriberline data link. A 512 Kbps bi-directional serial data port, which is clocked by SCL. SLD becomes a TTL compatible input when SDIR is high and an output capable of driving one TTL load when SDIR is low.
SIGX1 SIGX2 SIGX3	Transmit signaling inputs. Data present at SIGX(n) is latched by an internal signal preceding the falling edge of SDIR and is serially transferred on SLD during the transmit signaling byte. TTL compatible.
SIGR1 SIGR2 SIGR3	Receive signaling outputs. Data received serially on SLD during the receive signaling byte is latched on these outputs during the following byte. Capable of driving one TTL load.
SIGA SIGB SIGC SIGD	Programmable signaling pins. If the appropriate bit in the feature control memory is set high (either SIGDA, SIGDB, SIGDC, or SIGDD), the corresponding pin will become a receive signaling output, like SIGR(n). If the bit in the feature control memory is set low, the corresponding pin will become a transmit signaling input, like SIGX(n). Inputs will accept TTL level inputs, and outputs can drive one TTL load.

FUNCTIONAL DESCRIPTION

The 29C51 is a combined channel filter and PCM codec for use on analog line interface circuit boards in a digital telecommunications switching system. This device resides between the circuitry which provides the "BORSHT" functions for a given line, and the shared line card controller (LCC). It provides the transmit and receive voice-path filtering and companded analog-to-digital and digital-to-analog conversions necessary to interface a full duplex (4-wire) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system.

There are 10 signaling channels available on the 29C51 configured as three transmit, three receive, and four programmable for either direction. In addition, the 29C51 supports two secondary analog inputs and one secondary analog output. The 29C51 can alternatively use the secondary analog data received from the SLD interface to support three-party conferencing on chip.

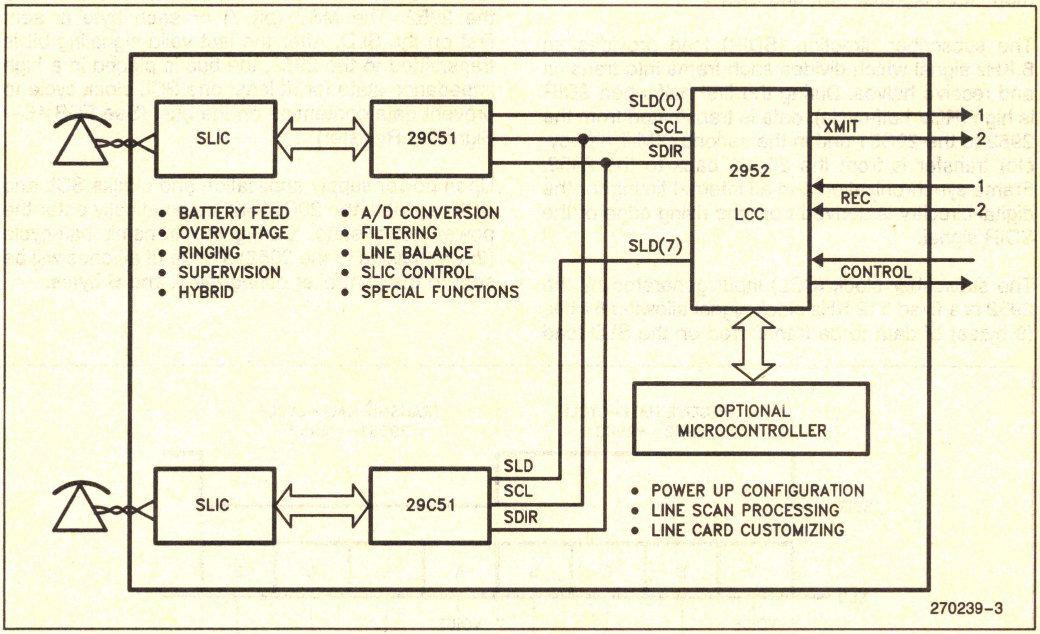


Figure 3. Analog Linecard

SLD Interface

The 29C51 is intended for use with the 2952 Line Card Controller which manages the transfer of all voice, feature control and signaling data to and from the Feature Control Combo and the system backplane. The interface between the two consists of just three leads, two of which are clock signals and the third a unique serial bus for communication. Up to eight 29C51 Feature Control Combos per line card can be controlled by one 2952, all sharing common clock signals, SCL and SDIR.

The subscriber direction (SDIR) lead provides an 8 KHz signal which divides each frame into transmit and receive halves. During the first half when SDIR is high (RCV half-cycle), data is transmitted from the 2952 to the 29C51 and in the second (XMIT half-cycle) transfer is from the 29C51 back to the 2952. Frame synchronization and all internal timing for the digital circuitry is derived from the rising edge of the SDIR signal.

The subscriber clock (SCL) input generated by the 2952 is a fixed 512 KHz clock signal allowing 64 bits (8 bytes) of data to be transferred on the SLD lead during each 125 μs frame. Depending on 2952 master clock frequency, the SCL duty cycle can be either 50% or 33%.

The subscriber line data link (SLD) is a bidirectional serial bus that transfers eight bytes of serial data to and from the 29C51 each frame. During the first half of each frame, RCV channel information is transferred to the 29C51 in four bytes consisting of voice, secondary analog, feature control, and signaling information. Similarly during the second half-cycle, four bytes of XMIT channel information are sent to the 2952. The MSB (bit 7) of each byte is sent first on the SLD. After the last valid signaling bit is transmitted to the 2952, the bus is placed in a high impedance state for at least one SCL clock cycle to prevent data contention on the bus. (See FCB#6—Signaling Register)

Upon power supply application and clocks SCL and SDIR applied, the 29C51 will automatically enter the power down state. During the transmit half-cycle (29C51 talking to the 2952) a code of all ones will be sent to the controller during the A and B bytes.

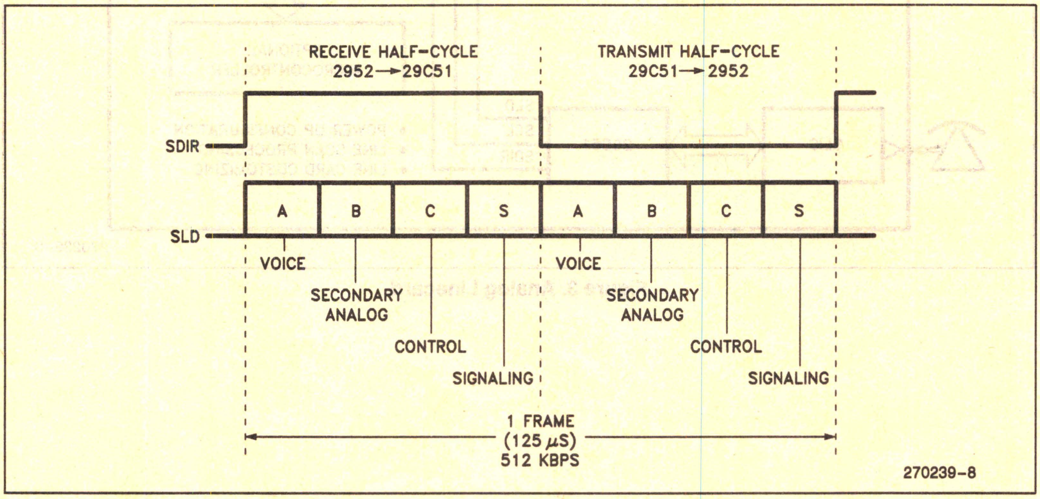


Figure 4. 29C51/2952 Interface

TRANSMIT AND RECEIVE OPERATION

Transmit Filter

A low pass anti-aliasing section is included on chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. The 29C51 specifications meet the digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 10.

A high pass section configuration rejects low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Gain of up to 20 dB can be set without degrading the performance of the filter.

Encoding

The output of the transmit filter is internally sampled by the encoder and held on an internal sample and hold capacitor. DC offset is corrected by an on-chip auto zero circuit. The signal is then encoded and presented as PCM data on the SLD lead on the first 8 bits of the XMIT half-frame (fifth byte). Secondary analog input signals are routed directly to the encoder and output in the sixth byte on the SLD.

Decoding

The PCM words received on the SLD are demultiplexed and sent to the decoder. The decoded value is held on an internal sample and hold capacitor. If the secondary analog channel is being used, the PCM word received in the second byte on the SLD is decoded, then held on another sample and hold capacitor before appearing on the secondary analog output (SAO). If, however, conferencing has been selected, the two converted signals will be added and subsequently passed to the receive filter.

Receive Filter

The receive section of the filter provides a passband flatness and stopband rejection which fulfills the AT&T D3/D4 specification and the CCITT G.712 recommendation. The receive filter transfer characteristics and specifications will be within the limits shown in Figure 11.

GENERAL OPERATION

External Gain Setting

Both transmit and receive gain levels are factory trimmed, but can be modified by external resistors during line card assembly. The value of transmit gain is adjusted by connecting resistors RT1 and RT2 (See Figure 5) at the two external gain setting control pins, TG1 and TG2. These two pins are the input and output of an on-board gain amplifier stage, and the resistors provide the necessary input and feedback for gain control. The value of external gain is given by:

$$A = 1 + RT1/RT2$$

For unity gain, pins TG1 and TG2 are tied together. A DC path must be provided for TG1. Similarly, for the receive section, external resistors RR1 and RR2 at pins VFR+, GSR, and VFR− set the external gain given by:

$$A = (RR1 + RR2)/(RR1 + 3RR2)$$

A value greater than 10 KΩ and less than 100 KΩ for R1 + R2 is recommended. The output is capable of driving loads of 300Ω at 3.2 Vp single ended or 600Ω at 6.4 Vp differentially.

Three additional gain settings of 0 dB, −6 dB and −9.54 dB can be realized without using any external components by strapping pin GSR to VFR−, GNDA, and VFR+, respectively.

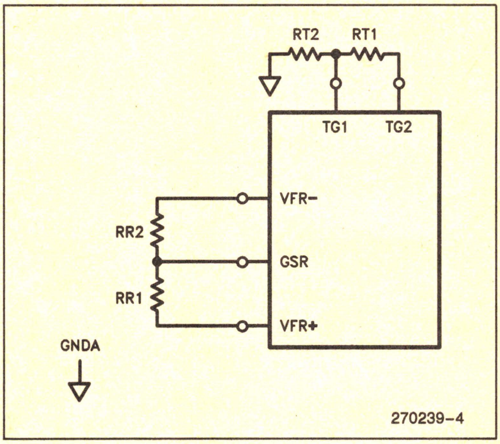


Figure 5. External Gain Connections

Hybrid Balancing Network

The 2- to 4-wire conversion necessary for subscriber interface is partially integrated on-chip. Network line balancing needed to minimize the trans-hybrid loss from the receive to transmit direction analog signals is handled internally. The three internal networks shown in Figure 7 may be selected by programming the appropriate feature control byte. These networks are integrated in a switched capacitor configuration and have single pole-zero characteristics in the 200 Hz to 3200 Hz range. They were chosen to serve a wide base of U.S. and European requirements, and can be used as standard line balancing networks or as test networks.

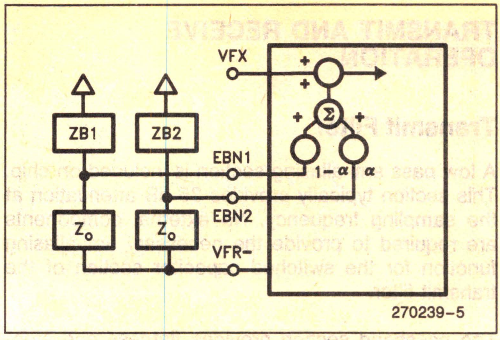


Figure 6. External Balance Network and Interpolation Configuration

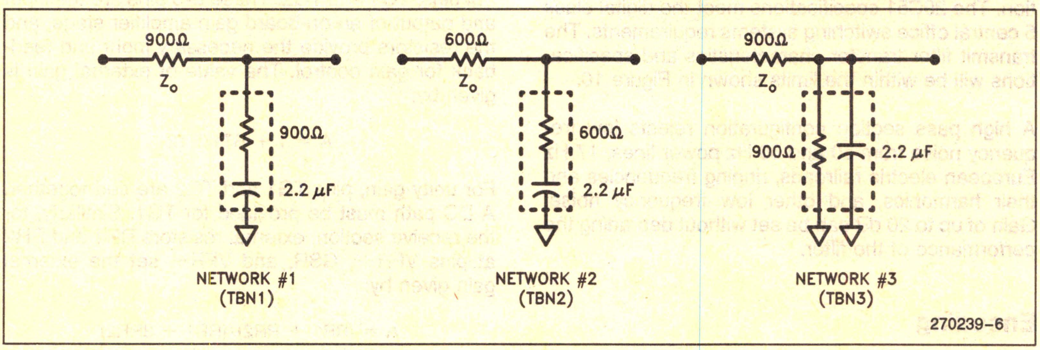


Figure 7. Internal Balance Networks

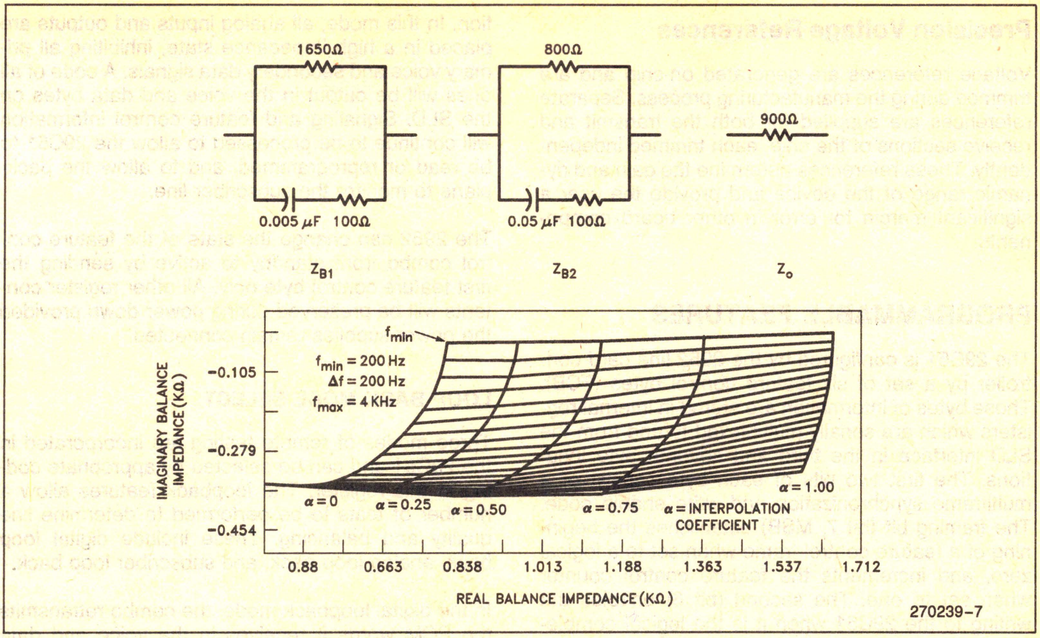


Figure 8. Typical External Balance Networks and Complex Impedance Plot

Additionally, the user may apply two external balance networks to accommodate varying subscriber loop characteristics (See Figure 6 for external connections). Presumably, these two networks can represent the two extremes of line conditions in different applications such as long or short loops and loaded or unloaded lines. To serve typical lines with characteristics in between the two extremes, the interpolation capability provides a weighted average of the network frequency characteristics. If the external network at EBN1 produces a transfer function H1(f) = ZB1 (Zo + ZB1) and the network at EBN2 produces H2(f) = ZB2 (Zo + ZB2), the balance signal can be programmed to have the transfer function H(f):

$$H(f) = \alpha\, H1(f) + (1-\alpha)\, H2(f)$$

Where "α" is the interpolation coefficient programmed to have any of the five values of 0, 0.25, 0.50, 0.75, or 1.0. Figure 6 displays how the subtraction of the coupling signal is implemented inside the device.

As an example, the two external networks shown in Figure 8 represent typical hybrid balance networks for loaded (ZB1) and unloaded (ZB2) analog loops. The graph in Figure 8 shows the real and imaginary components of the equivalent impedance of these two networks as a function of frequency and the interpolation coefficient. A DC path must be provided at EBN1 and EBN2 when in use.

Secondary Analog Channel/ Conferencing

The 29C51 offers two simultaneous unfiltered information channels beyond the primary channel. Narrow band analog signals can be supplied for such applications as telemetry, teleconferencing, remote loop testing, or various control uses.

The secondary analog channel is accessed through two inputs, SAI1 and SAI2, sampled either single ended or differentially. A DC path must be provided at SAI1 and SAI2 when in use. The unfiltered secondary analog output, SAO, is a stair-step signal with the inherent sinx/x frequency rolloff following D/A conversion.

To allow three-party conferencing, the third party voice information can be transmitted and received during the data bytes carrying the secondary analog channel information. The primary and secondary voice signals are held on separate internal capacitors following D/A conversion, then passed through a −3 dB attenuator and summed together. The combined signal is smoothed in the receive filter and passed onto the output power amplifier.

Precision Voltage References

Voltage references are generated on-chip and are trimmed during the manufacturing process. Separate references are supplied for both the transmit and receive sections of the chip, each trimmed independently. These references determine the gain and dynamic range of the device and provide the user a significant margin for error in other board components.

PROGRAMMABLE FEATURES

The 29C51 is configured by the 2952 line card controller by a set of six feature control bytes (FCB). These bytes of information are stored in internal registers which are serially multiplexed to and from the SLD interface in the third and seventh byte locations. The first two bits of each byte consist of a multiframe synchronization and write enable code. The framing bit (bit 7, MSB) establishes the beginning of a feature control frame when set to a logical zero, and increments the feature control counter when set to one. The second (bit 6) enables the writing to the 29C51 when it is the logical complement of the framing bit.

When writing new feature control information to the 29C51, the first byte should contain a framing (F) and write enable (WE) header of 01 (F = 0 and WE = 1). This designates a new frame of information to transfer. The subsequent bytes should each have F = 1 to advance the counter, and WE = 0 to enable the write operation.

The controller can also request to verify the feature control register contents by sending a 00 or 11 at the beginning of the byte to be read. To read the first byte, a 00 F/WE code should be sent while each subsequent byte should have a 11 header. An internal six-stage counter is set on the first byte verified then incremented once each 125 μs frame. It is reset only upon detection of a 01 or 00 F/WE. Once the counter is greater than six, neither read nor write modes may be selected by sending the 29C51 a 11 framing and write enable code. The 29C51 will then echo in byte 7 the data it received in byte 3.

FCB #1—Power Up/Down, Loop Back Mode, μ/A-Law Select Register

POWER UP AND DOWN

The 29C51 can be instructed to go into the power down or standby mode for reduced power consumption. In this mode, all analog inputs and outputs are placed in a high impedance state, inhibiting all primary voice and secondary data signals. A code of all ones will be output in the voice and data bytes on the SLD. Signaling and feature control information will continue to be processed to allow the 29C51 to be read or reprogrammed, and to allow the backplane to monitor the subscriber line.

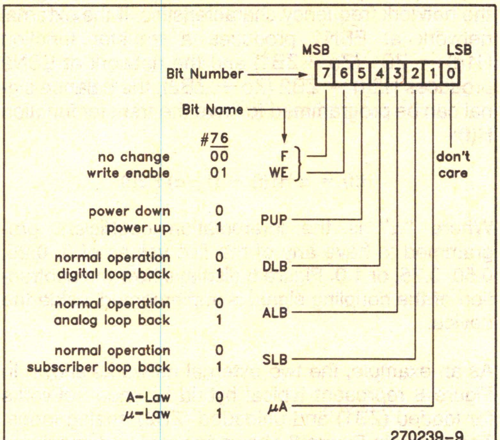
The 2952 can change the state of the feature control combo from standby to active by sending the first feature control byte only. All other register contents will be preserved during power down provided the power supplies remain connected.

LOOP BACK MODE SELECT

Three modes of remote testing are incorporated in the 29C51 and can be selected by appropriate coding in this register. The loopback features allow a number of tests to be performed to determine line quality and balancing. These include digital loop back, analog loop back, and subscriber loop back.

In the digital loopback mode, the combo retransmits the PCM words it receives in the voice and data bytes of the SLD back to the line card controller in the same frame. This feature allows path verification and testing of the circuit up to the slave device.

When the analog loopback mode is selected, the analog output VFR+ is internally connected to the analog input VFX. This feature allows functional testing of the combo as well as gain adjustment. The secondary analog channel is unaffected during this operation.



270239-9

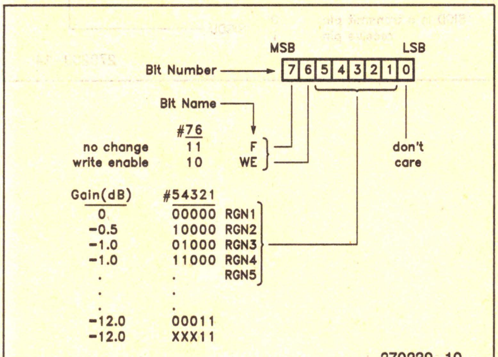
In the third test mode, subscriber loopback, the digital output of the A/D converter is internally connected to the input of the D/A converter. The analog signal input to VFX is sent through the transmit filter, encoded, then decoded, filtered and output to VFR+ and VFR−. This mode is used primarily for simplifying analog to analog testing from the subscriber side of the line card. If the secondary analog inputs and output are being used, they will be looped back in the same manner.

CONVERSION LAWS

The 29C51 can be selected for either μ-law or A-law operations. A user can select either conversion law by assigning the corresponding bit. A logical 1 in bit 1 would select μ-law while a logical 0 would select A-law conversions. Both conversions follow CCITT recommendation G.711.

FCB #2—Receive Programmable Gain Register

The receive gain levels can be adjusted by applying external resistors as mentioned earlier, or by selective programming of this register. A range from 0 dB to −12 dB in 0.5 dB increments can be realized for the receive channel.



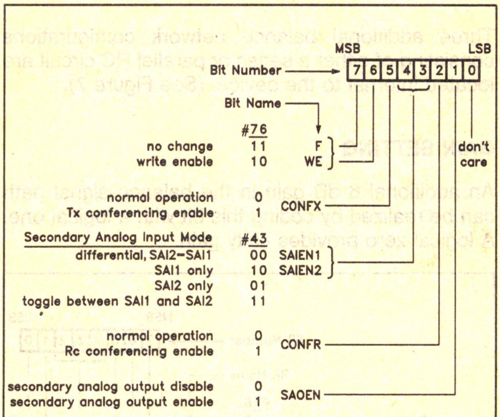
270239-10

FCB #3—Secondary Analog Channel Register

SECONDARY ANALOG INPUTS AND OUTPUTS

The two inputs to the secondary analog channel, SAI1 and SAI2, can be programmed to be encoded either single ended or differentially. An analog signal applied at the selected input may be encoded once every 125 μs in addition to the primary voice channel. Alternatively, both SAI1 and SAI2 may be selected in which case each signal would be encoded in alternating frames at an effective sampling rate of 4 KHz. The LSB of the encoded word would toggle between 0 and 1 to designate which input it was encoded from. A "1" in the LSB represents SAI1 and a "0" for SAI2. The two inputs may also be used in a differential mode, resulting in SAI2 subtracted from SAI1.

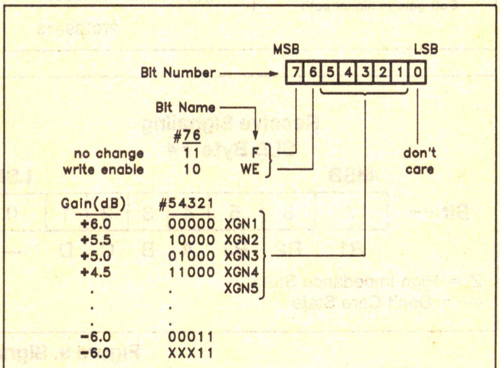
The receive section of the secondary analog channel can be programmed to direct the data byte output onto SAO, or to add the analog signal to the primary voice channel for conferencing.



270239-11

FCB #4—Transmit Programmable Gain Register

The gain setting of the transmit section of the chip operates in the same manner as the receive gain register. A 12 dB range from −6.0 dB to + 6.0 dB in 0.5 dB increments is available.



270239-12

FCB #5—Balance Network Select and Gain Register

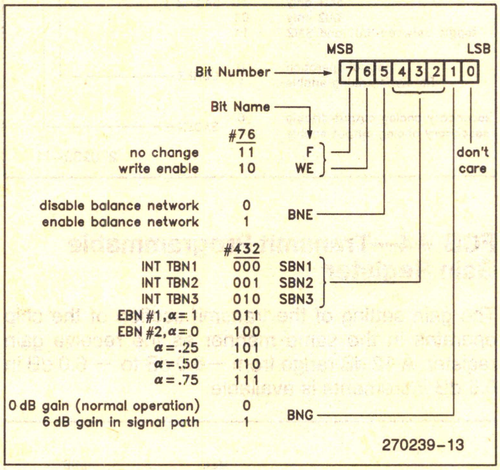
BALANCE NETWORKS

The 29C51 offers a choice of internal or external hybrid balancing. Externally, two balance networks connected to pins EBN1 and EBN2 can be used independently, or as a weighted average of the two. The weighting factor, or interpolation coefficient, can range from 0 to 1 in steps of 0.25. Setting "α" to be 1 or 0 results in selecting either EBN1 or EBN2 respectively.

Three additional balance network configurations consisting of either a series or parallel RC circuit are located internal to the device. (See Figure 7).

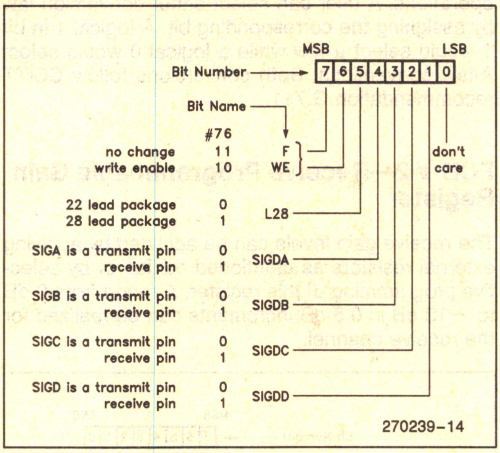
GAIN SETTING

An additional 6 dB gain in the balance signal path can be realized by coding this bit with a logical one. A logical zero provides unity gain.



FCB #6—Signaling Register

Four pins are provided on both the 29C50 and 29C51 chips to be used as selectable transmit or receive signaling inputs. A code of one in the respective bit commits the pin to receive signal information and a zero to transmit. The signaling field format as it appears on the SLD bus is shown in Figure 9 for the 29C51. R1, R2 and R3 correspond to signaling information received on SIGR1, SIGR2, and SIGR3 respectively. Similarly, programmable pins SIGA, SIGB, SIGC, SIGD, and transmit pins SIGX1, SIGX2, and SIGX3 are coded into the bit location as shown below.



Receive Signaling
SLD Byte #4

	MSB							LSB
Bit →	7	6	5	4	3	2	1	0
	R1	R2	R3	A	B	C	D	—

Transmit Signaling
SLD Byte #8

	MSB							LSB
Bit →	7	6	5	4	3	2	1	0
	X1	X2	X3	A	B	C	D	Z

Z = High Impedance State
— = Don't Care State

Figure 9. Signaling Field Format

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias .... $-10°C$ to $+80°C$
Storage Temperature .... $-65°C$ to $+150°C$
All Input and Output Voltages with Respect to V_{BB} .... $-0.3V$ to $+13V$
All Input and Output Voltages with Respect to V_{CC} .... $-13V$ to $+0.3V$
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

$T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 5\%$, $V_{BB} = -5V \pm 5\%$; SCL (50% duty), SDIR, SLD applied GNDD = 0V, GNDA = 0V. Typical values are for $T_A = 25°C$ and nominal power supply values.

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current	−0.3		±10	μA	$V_{BB} \leq V_{IN} \leq V_{CC}$
V_{IL}	Input Low Voltage			0.8	V	
V_{IH}	Input High Voltage	2.2		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	$I_{OL} \geq -1.6$ mA, 1 TTL Load
V_{OH}	Output High Voltage	2.4			V	$I_{OH} \leq 50$ μA, 1 TTL Load

POWER DISSIPATION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{CCL}	V_{CC} Operating Current		9		mA	RCV Signaling Byte = 0
I_{BBL}	V_{BB} Operating Current		9		mA	RCV Signaling Byte = 0
I_{CCO}	V_{CC} Standby Current		0.8		mA	
I_{BBO}	V_{BB} Standby Current		0.8		mA	
P_{D0}	Standby Power Dissipation		8		mW	
P_{D1}	Operating Power Dissipation		90		mW	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

TG1 = TG2, Transmit Programmable Gain = 6 dB; GSR = VFR−, Receive Programmable Gain = 0 dB

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response Tolerance		±0.15		dB	Signal Input of 0 dBm0 f = 1.02 KHz
DmW	Digital Milliwatt Response Tolerance		±0.15		dB	f = 1.02 KHz
$DmW_{\mu V}$	Digital Milliwatt Response VFR+, μ-law		6.14 1.571		dBm Vrms	VFR+ Single-Ended Output $R_L = 600\Omega$ Signal Input per CCITT G.711

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS (Continued)

TG1 = TG2, Transmit Programmable Gain = 6 dB; GSR = VFR−, Receive Programmable Gain = 0 dB

GAIN AND DYNAMIC RANGE (Continued)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
DmW_{AV}	Digital Milliwatt Response VFR+, A-law		6.17 1.576		dBm Vrms	VFR + Single-Ended Output R_L = 600Ω Signal Input per CCITT G.711
$DmW_{\mu s}$	Digital Milliwatt Response at SAO, μ-law		3.70		dBVrms	No Load; No Sin x/x Correction
DmW_{AS}	Digital Milliwatt Response at SAO, A-law		3.73		dBVrms	No Load; No Sin x/x Correction
$OTLP_{\mu X}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.116 0.785		dBm Vrms	μ-law, R_L = 600Ω
$OTLP_{AX}$	Zero Transmission Level Point Transmit Channel (0 dBm0)		0.149 0.788		dBm Vrms	A-law, R_L = 600Ω

GAIN TRACKING

Reference level = 0 dBm0 at 1.02 KHz, TG1 = TG2, GSR = VFR−, Transmit Programmable Gain = 6 dB, Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
GT_T	Transmit Gain Tracking Error Sinusoidal Input; μ- or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
GT_R	Receive Gain Tracking Error Sinusoidal Input; μ- or A-law		±0.25 ±0.50 ±1.2		dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 AT&T PUB43801 and CCITT G.712—Method 2

ANALOG INTERFACE, RECEIVE PRIMARY AND SECONDARY CHANNELS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
R_{OR}	Output Resistance, VFR+ VFR−		1		Ω	
V_{OSR1}	Output Offset, VFR+ or VFR−, Single Ended		50		mV	Relative to GNDA
V_{OSR2}	Output Offset, VFR+ to VFR−, Differential		75		mV	
C_{LR}	Load Capacitance, VFR+, VFR−			100	pF	
V_{OR1}	Max Output Voltage Swing Across R_L, VFR+, VFR−, Single-Ended Connection	±3.2			Vp	$R_L \geq 300\Omega$
V_{OR2}	Max Differential Output Voltage Swing, VFR+, VFR−	±6.4			Vp	$R_L \geq 600\Omega$
P_{OR}	Differential Output Power, VFR+, VFR−			15.3	dBm	$R_L = 600\Omega$
R_{ORS}	Output Resistance, SAO		25		Ω	
V_{OSR}	Output Offset, SAO		50		mV	
C_{LRS}	Load Capacitance, SAO			20	pF	
R_{LRS}	Load Resistance, SAO	10			KΩ	
V_{ORS}	Output Voltage Swing, SAO	±3.2			Vp	$R_L \geq 10$ KΩ

ANALOG INTERFACE, TRANSMIT PRIMARY AND SECONDARY CHANNELS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{BX}	Input Leakage Current, EBN1, EBN2, TG1		100		nA	$-1.6V < VFX < 1.6V$
R_{IX1}	Input Resistance, VFX		500		KΩ	$-1.6V < VFX < 1.6V$
R_{IX2}	Input Resistance, EBN1, EBN2, TG1		10		MΩ	$\pm 1.6V < VFX < 1.6V$
$CMRR_S$	Common Mode Rejection, SAI1, SAI2		40		dB	Differential SAI Conversion
TGmax	Max Transmit Gain Adjust			20	dB	
V_{OTG}	Max Output Voltage Swing TG2			±3.2	V	$R_L \geq 10$ KΩ, (Note 1)
C_{LX}	Load Capacitance, TG2		20		pF	
R_{LX}	Load Resistance, TG2	10			KΩ	

NOTE:
1. Transmit programmable gain must be set at −6 dB to encode this level without clipping.

DISTORTION (Primary Channel)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
SD_X, SD_R	Signal to Distortion, μ or A-law Sinusoidal Input; CCITT G.712— Method 2 Half Channel	35 29 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X, DP_R	Single Frequency Distortion Products In Band (2nd or 3rd Harmonic Half Channel)		−50	−47	dB	Input = 1.02 KHz 0dBm0 AT&T Advisory #64 (3.8)
IMD_1	Intermodulation Distortion, End to End Measurement			−40	dBm0	CCITT G.712(7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−50	dBm0	CCITT G.712(7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−27	dBm0	CCITT G.712(6.1)
SIS	Spurious In Band Signals, End to End Measurement			−40	dBm0	CCITT G.712(9)
D_{AX}	Transmit Absolute Delay		180		μs	0 dBm0, 1.02 KHz Includes Delay Through A/D
D_{DX}	Transmit Differential Envelope Delay; Relative to Minimum Envelope Delay (1.4 KHz)		170 95 45 105		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz
D_{AR}	Receive Absolute Delay		125		μs	0 dBm0, 1.02 KHz Includes Delay Through D/A
D_{DR}	Receive Differential Envelope Delay; Relative to Minimum Envelope Delay (300 Hz)		45 35 85 110		μs μs μs μs	f = 500–600 Hz f = 600–1000 Hz f = 1000–2600 Hz f = 2600–2800 Hz

NOISE (Primary Channel)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted		12		dBrnCO	Transmit Gain Adjust = 0 dB
N_{XP1}	Transmit Noise, Psophometrically Weighted		−78		dBmOp	Transmit Gain Adjust = 0 dB
N_{RC1}	Receive Noise, C-Message Weighted		10		dBrnCO	Unity Gain; Idle Code
N_{RP1}	Receive Noise, Psophometrically Weighted		−80		dBmOp	Unity Gain; Idle Code
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit or Receive Channel		−35		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).
$PSRR_2$	V_{BB} Power Supply Rejection Transmit or Receive Channel		−30		dB	Idle Channel; 200 mV P-P Signal on Supply DC to 50 KHz; (Note 1).

CROSSTALK

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
CT_{TR}	Crosstalk, Transmit Primary Voice to Receive Primary Voice		−75		dB	Input = 0 dBm0, Unity Gain 1.02 KHz; Idle Code on SLD Voice and Data Bytes
CT_{RT}	Crosstalk, Receive Primary Voice to Transmit Primary Voice		−75		dB	0 dBm0, 1.02 KHz Signal at SLD Receive Voice Byte; VFX = GNDA; Secondary Channels Off
CT_{ST}	Crosstalk, Transmit Secondary Channels to Transmit Primary Voice		−70		dB	SAI = 0 dBm0, 1.02 KHz; VFX = GNDA Idle Code on SLD Voice and Data Bytes
CT_{SR}	Crosstalk, Receive Secondary Channel to Receive Primary Voice		−70		dB	0 dBm0, 1.0-2 KHz at SLD Data Byte VFX = SAI = GNDA

NOTE:
1. Measured at SLD Voice bytes for transmit channel. Measured at $V_{FR}+$ for receive channel.

TRANSMIT VOICE FREQUENCY CHARACTERISTICS

TG1 = TG2, Transmit Programmable Gain = 6 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at VFX
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−22	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.70		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	
ΔG_{PX}	Programmable Gain (Cumulative Error)		±0.25		dB	f = 1.02 KHz for all Steps

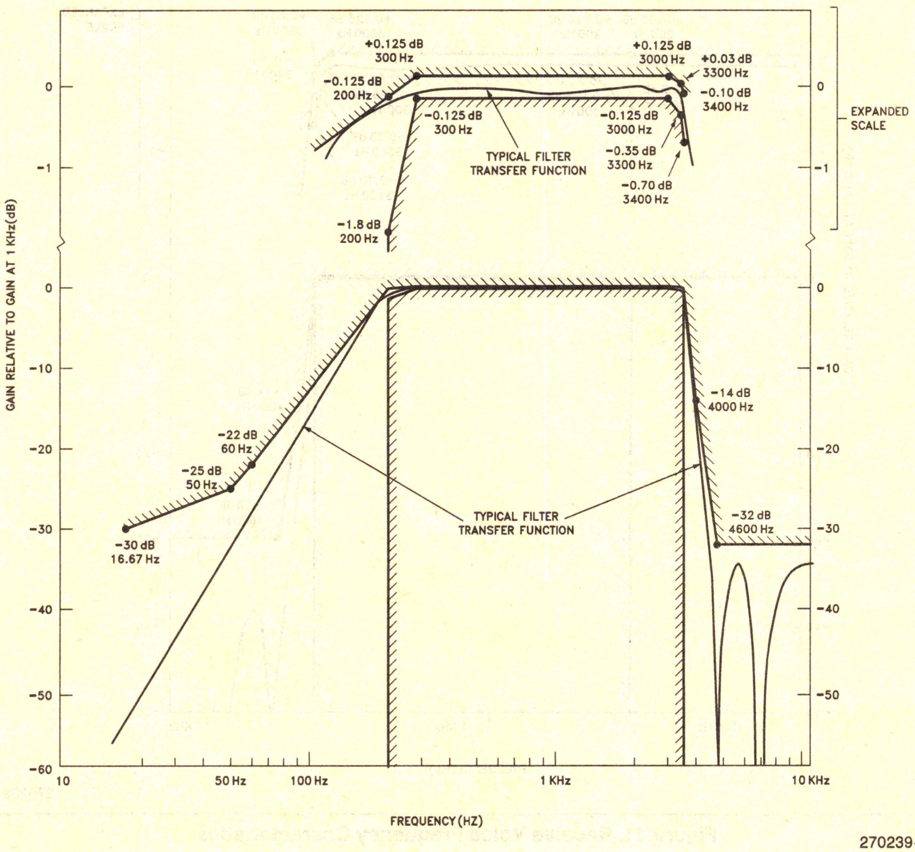


Figure 10. Transmit Voice Frequency Characteristics

RECEIVE VOICE FREQUENCY CHARACTERISTICS

GSR = VFR−, Receive Programmable Gain = 0 dB

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Input on SLD
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.70		−0.1	dB	
	4000 HZ			−14	dB	
	4600 Hz and Above			−30	dB	
ΔG_{PR}	Programmable Gain Accuracy (Cumulative Error)		±0.25		dB	f = 1.02 KHz for all Steps

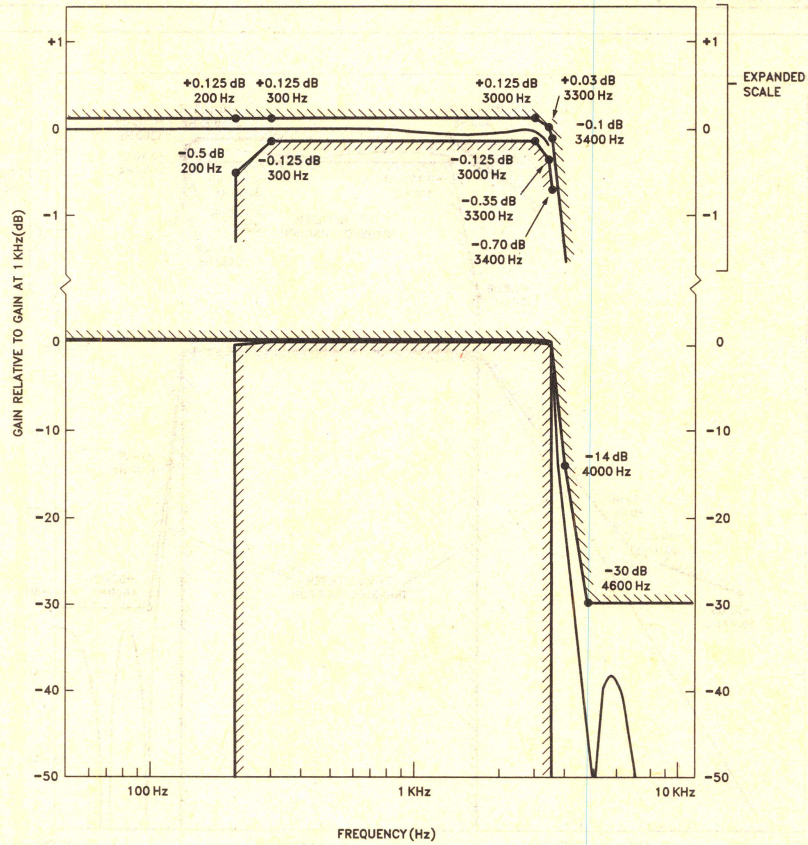


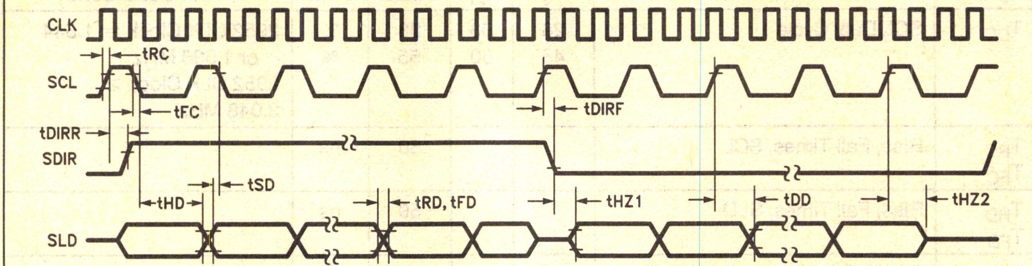
Figure 11. Receive Voice Frequency Characteristics

A.C. CHARACTERISTICS—TIMING PARAMETERS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
T_{DC}	SCL Duty Cycle	28 45	33 50	38 55	% %	2952 CLK Clock = 1.544 or 1.536 MHz 2952 CLK Clock ≥ 2.048 MHz
T_{RC} T_{FC}	Rise, Fall Times, SCL			50	ns	
T_{RD} T_{FD}	Rise, Fall Times, SLD			50	ns	
T_{DIRR}	SCL to SDIR Rising Edge Delay	−100		100	ns	
T_{DIRF}	SCL to SDIR Falling Edge Delay	−100		+420	ns	
T_{DD}	SCL to SLD Delay	0		200	ns	29C51 Transmitting (Note 1)
T_{SD}	Set-up Time, SLD to SCL	100			ns	2952 Transmitting
T_{HD}	Hold Time, SCL to SLD	100			ns	
T_{HZ1}	SDIR to SLD Active	0		100	ns	Byte 1, Bit 1 29C51 Transmitting
T_{HZ2}	SCL to SLD High Impedence	0		50	ns	After last SIGX Bit
T_{SS}	Set-up Time, Signaling Inputs to SLD Byte #3, Bit 0	1			μs	
T_{HS}	Hold Time, SLD Byte 4 Bit 7 for all Signaling Inputs	1			μs	
T_{DS}	Delay SLD Byte 5 to Signaling Outputs			1	μs	

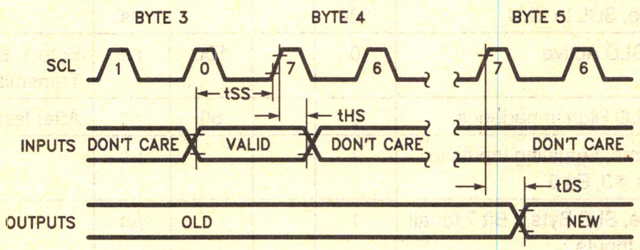
NOTE:
1. In cases where T_{DIRF} is positive, T_{DD} is measured from the SDIR edges.

TIMING PARAMETERS



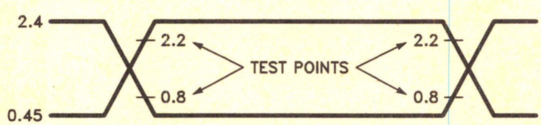
270239-17

SIGNALING TIMING



270239-18

A.C. TESTING INPUT, OUTPUT WAVEFORM



270239-19

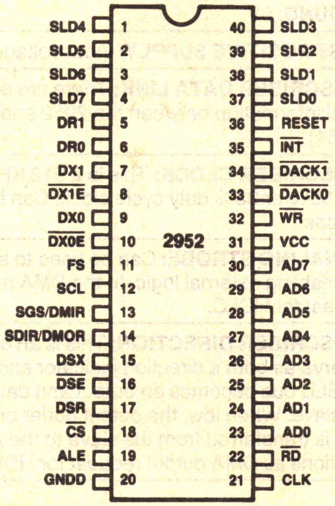
A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.2V for a Logic "1" and 0.8V for a Logic "0".

iATC 2952 INTEGRATED LINE CARD CONTROLLER

- **Provides Complete Backplane Interface for 8 to 16 Subscribers**
- **Performs All Timeslot Assignments**
- **2 Full-Duplex, Serial TDM Highways**
- **Serial, Bidirectional Packetized Highway for Signaling Control**
- **Standard MCS® μP Interface with Two Channel DMA and Interrupt**
- **Implements HDLC Protocol to Guarantee Integrity of All Signaling and Control Information**
- **Supports Four Control Options Local or Global Microprocessor Direct or Interleaved HDLC Control**
- **Designed for 24, 32, 48 or 64 Timeslot Systems**
- **Common Backplane Interface Supports ISDN Upgradability**

The Intel iATC 2952 Line Card Controller (LCC) is a special purpose I/O controller optimized for use in all types of telecommunication switching systems. The 2952 is intended for use with up to sixteen subscriber devices in both analog and digital line circuits. It is also useful as a general purpose I/O controller for other applications.

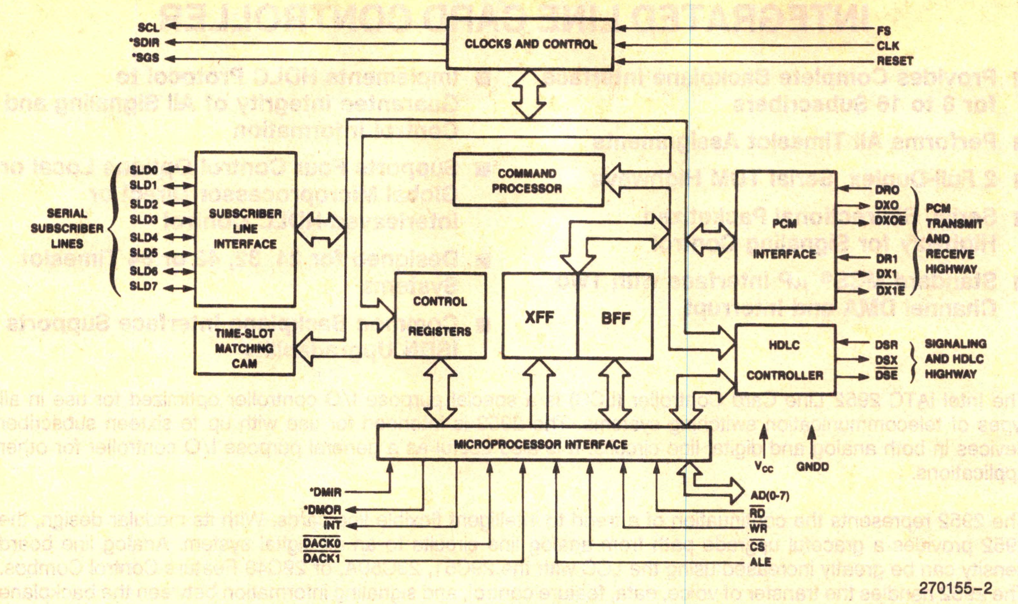
The 2952 represents the continuation of a trend to intelligent flexible line cards. With its modular design, the 2952 provides a graceful upgrade path from analog line circuits to an all digital system. Analog line board density can be greatly increased using the LCC with the 29C51, 29C50A, or 29C48 Feature Control Combos. The 2952 handles the transfer of voice, data, feature control, and signaling information between the backplane and up to 8 29C51's, 16 29C50A's, or 16 29C48's. The 2952 will interface with and control all Intel SLD compatible slave devices. The 2952 emphasizes highly serial interfaces, thus reducing the number of digital interconnections both to the subscriber device and to the backplane.



270155–1

Figure 1. Pin Configuration

November 1986
Order Number: 270155-001



NOTE:
*Dual Function Pin.

Figure 2. 2952 Block Diagram

Table 1. Pin Description

Symbol	Pin No.	Function
GNDD	20	**GROUND:** 0V.
V_{CC}	31	**MOST POSITIVE SUPPLY:** Input voltage is +5V ±5%.
SLD0–7	37–40 1–4	**SUBSCRIBER DATA LINK:** There are eight bidirectional pins that transfer serial information between the 2952 and the subscriber devices (e.g. 29C51).
SCL	12	**SUBSCRIBER CLOCK:** This is a 512 KHz signal generated by the 2952 with 50% or 33% duty cycle clock. Can be connected up to 8 slave devices.
SGS/DMIR	13	**SIGNALING STROBE:** Can be used to strobe signaling bits or voice bytes for enabling external logic. In the DMA mode DMIR functions as DMA input request for HDLC.
SDIR/DMOR	14	**SUBSCRIBER DIRECTION:** This is an 8 KHz signal generated by the 2952 to serve as both a direction indicator and a slave frame sync. When high, the SLD bus becomes an output and data is transferred from the 2952 to the slave. When low, the output buffer on the slave SLD pin is enabled and data is transferred from the slave to the 2952. In the DMA mode, DMOR functions as DMA output request for HDLC.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
$\overline{CS}$	18	**CHIP SELECT:** Enables $\overline{RD}$ or $\overline{WR}$. A low level at this input allows the 2952 to accept commands or data from a microprocessor within a write cycle, or to transmit data during a read cycle. When no local μP is connected, this pin should be connected to GNDD.
ALE	19	**ADDRESS LATCH ENABLE:** (Active high input.) On falling edge of this input signal, data on AD(0-7) is latched into the selected register. When no local μP is connected, this pin should be connected to GNDD.
$\overline{RD}$	22	**READ STROBE:** (Active low input.) When input is low, data is transferred from selected register on to μP bus. When no local μP is connected, this pin should be connected to GNDD.
AD(0-7)	23-30	**ADDRESS/DATA PINS:** Standard μP bus used to transfer address and data between the μP and internal registers of the 2952. When no local μP is connected, the unique ID is hardwired on pins AD(0-7).
$\overline{WR}$	32	**WRITE STROBE:** (Active low input.) When $\overline{WR}$ transitions from low to high, data on pins AD(0-7) are latched into the selected register. When no local μP is connected, this pin should be connected to GNDD.
$\overline{DACK0}$ $\overline{DACK1}$	33 34	**DMA ACKNOWLEDGE:** $\overline{DACK0}$ is used to acknowledge the DMA output, and $\overline{DACK1}$ is used for DMA input. When no local μP is connected, these pins are used to hardwire mode information.
$\overline{INT}$	35	**INTERRUPT REQUEST:** A standard μP interrupt, with active low output.
DR1	5	**RECEIVE PCM HIGHWAY 1:** Serial words are received on PCM highway 1 at this interface.
DR0	6	**RECEIVE PCM HIGHWAY 0:** Serial words are received on PCM highway 0 at this interface.
DX1	7	**TRANSMIT PCM HIGHWAY 1:** Serial words are transmitted onto PCM highway 1 at this interface.
$\overline{DX1E}$	8	**TRANSMIT PCM HIGHWAY 1 ENABLE:** Used to enable external tristate buffers to drive signals onto the PCM highway. The signal goes low while the 2952 is transmitting onto PCM highway 1.
DX0	9	**TRANSMIT PCM HIGHWAY 0:** Serial words are transmitted onto PCM highway 0 at this interface.
$\overline{DX0E}$	10	**TRANSMIT PCM HIGHWAY 0 ENABLE:** Used to enable external tristate buffers to drive signals onto the PCM highway. The signal goes low while the 2952 is transmitting onto PCM highway 0.
DSX	15	**TRANSMIT SIGNALING HIGHWAY:** Serial signaling and control data is transmitted on this dedicated HDLC highway.
$\overline{DSE}$	16	**TRANSMIT SIGNALING HIGHWAY ENABLE:** Used to enable external tristate buffers to drive signals onto the transmit signaling highway. The signal goes low while the 2952 is transmitting an HDLC packet onto DSX.
DSR	17	**RECEIVE SIGNALING HIGHWAY:** Serial signaling and control data is received on this dedicated HDLC highway.
FS	11	**FRAME SYNCHRONIZATION:** System sync pulse indicating beginning of a 125 μs frame.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
CLK	21	**MASTER CLOCK:** System input clock provides basic timing for the 2952 and is synchronous to the PCM clock. The clock rate determines the number of timeslots on the transmit and receive PCM highways, ranging from 24, 32, 48 or 64 per frame.
RESET	36	**RESET:** (Active high input.) When high, 2952 internal circuitry is reset. The minimum reset pulse must be 16 complete CLK clock cycles wide.

FUNCTIONAL DESCRIPTION

The 2952 is a highly integrated line card controller which concentrates and multiplexes all digital information that passes between a line card and the next switching or control level in a digital telecommunications system. It controls time switching functions between the individual subscriber line devices and the system backplane Time Division Multiplexed (TDM) highways. In addition, it manages the transfer of all signaling and control messages, either to an optional local microprocessor or to a central control processor. The 2952 implements all protocol control functions using the HDLC format for all information transmitted between the line card and the central processor.

EXTERNAL INTERFACE

The 2952 LCC supports interfaces for the subscriber line devices, an optional local microprocessor, and the backplane PCM and HDLC highways. Each is described briefly below.

Subscriber Line Interface

The LCC provides 8 serial, bidirectional ports for the digital transmission of voice, data, control, and signaling information to and from the subscriber. These leads, SLD0–SLD7, can be used to interface to both analog and digital line card subscribers (see Figure 3).

The Slave Clock SCL, is a fixed 512 KHz signal output used to transfer all signals between the subscriber device and the 2952. Data is received and transmitted upon the rising edge of SCL.

Data transmission direction is controlled by the Slave Direction clock, SDIR. This 8 KHz signal divides the frame transfer into transmit and receive halves referenced to the subscriber as shown in Figure 3. When SDIR is high, (RCV half-cycle), information is transmitted from the 2952 to the subscriber in four bytes consisting of voice, data, feature control, and signaling information. In the second half (XMIT-half cycle), the subscriber circuit sends four bytes of XMIT data back to the 2952.

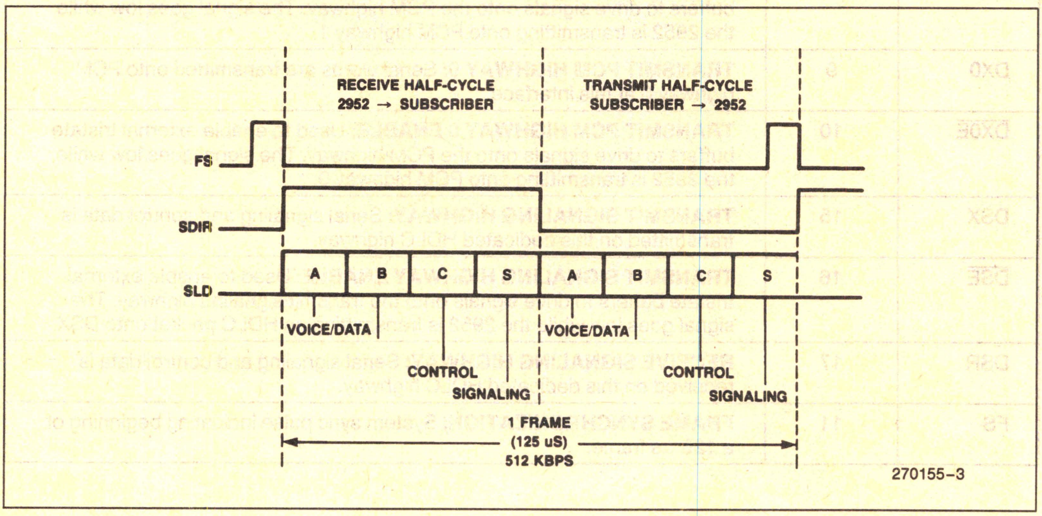


Figure 3. SLD Interface

Backplane Interface

The LCC supports dual TDM voice/data highways as well as a separate high speed serial highway for signaling and control information. This information is packetized and protected in HDLC format for transmission to a central processor.

The system clock (CLK) provides data rate transfers for both the TDM and HDLC links. The 2952 can operate in 24, 48, 32 or 64 timeslots systems. Any subscriber has access to any timeslot on either TDM highway. The 2952 allows the flexibility of programmable rising or falling edge latching of data onto the highways. Additionally, PCM highway delays can be compensated for by programming a phase shift in both transmit and receive timeslots as referenced to the frame synchronization pulse. The starting point of the bytes can be shifted up to 7 CLK clock cycles for both transmit and receive directions in half clock increments.

Microprocessor Interface

The microprocessor interface provides a communication path for a local μP and the backplane and/or slave devices. The 2952 is designed to operate with standard Intel 8-bit parallel microprocessors, such as the MCS®-48, MCS®-51, MCS®-85, and iAPX-86 families. Interrupt capability, direct-memory-access request and acknowledge signals and a full feature multiplexed address data bus are incorporated into this interface.

Alternatively, the 2952 can operate in a stand alone mode on the line card in systems using a more centralized processing architecture. In this mode, the individual line board address and initialization information is hardwired onto the microprocessor interface pins.

2952 BASE ARCHITECTURE

The 2952 can be partitioned into three functional blocks, according to the type of data transfers that each provides. The synchronous portion comprises the subscriber and PCM highway interfaces. Included in this section is also the Master Timing Unit, a CAM (Content Addressable Memory) for timeslot matching, a MODE register to configure the 2952 and for the determination of the type of HDLC data exchange, and the internal bus for a communication link between the various interfaces and registers. The PCM Interface Unit and the Subscriber Interface Unit with Last Look logic are also grouped into this segment. The Last Look logic monitors the status of signaling information received from each subscriber every frame. Any change of status is reported to the local μP or to the LCC bus control unit.

The asynchronous portion includes the local microprocessor interface and the serial HDLC signaling and control interface. The HDLC controller is compatible with ISO/CCITT recommendation X.25, and is designed for either point-to-multipoint configurations as a primary station, or in point-to-point as a secondary station. Each 2952 is accessed through an 8-bit address, allowing up to 255 secondaries to be addressed on one HDLC serial line. The logic level of the HDLC implementation and the distribution and compilation of the data packages are handled in a separate command unit contained in this portion.

The synchronous and asynchronous portions are linked to one another by a set of buffers and a control unit for the LCC internal bus. Two 16-byte by 8-bit FIFO's are used for intermediate storage of messages. The XFF, or Transmit FIFO buffers data packages for transmission to the central processor through the HDLC interface. The type of data loaded is either from the Last Look logic or from the μP in package form with direct addressing to the 2952. The BFF, or bidirectional FIFO, is used for data exchange between the central controller (via the HDLC interface), the local microprocessor (via the μP bus), and the LCC (via the LCC bus).

MODES OF OPERATION

The 2952 may operate in either a primary or secondary command mode within a single system. When instructed as a primary station, a local microprocessor must be used to instruct the 2952 and to generate control messages for other stations. This mode is used primarily by unit or group controllers to command secondary 2952's. When in the secondary mode, the 2952 executes received HDLC commands from the group controller. Additionally, a transparent command mode may be configured in which all HDLC messages received from the backplane are passed directly to the local microprocessor. This allows a secondary 2952 to execute user defined protocol and commands.

The 2952 can operate in one of two HDLC communication modes—dedicated HDLC or interleaved HDLC. In the dedicated configuration, HDLC messages are received on DSR and are transmitted on DSX. The interleaved mode reserves up to two timeslots per frame for transmission of signaling and control messages on the PCM highways. The HDLC packets are disassembled and interleaved into programmed timeslots on either of the two highways. Alternatively, the microprocessor can communicate directly to the central controller via a direct connection, bypassing the 2952 HDLC interface completely.

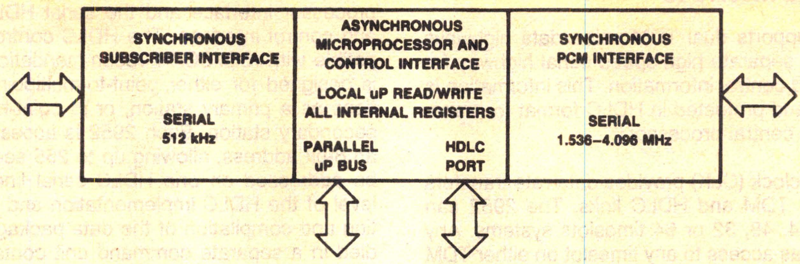


Figure 4. Architectural Diagram

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C

Storage Temperature .... −65°C to +125°C

All Input and Output Voltages
with Respect to GNDD .... −0.3V to +7V

Total Power Dissipation 1.5W

******Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, GNDD = 0V
Typical values are for T_A = 25°C and nominal power supply value

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current	−10		+10	μA	GND ≤ V_{IN} ≤ V_{CC}
I_{OL}	Output Leakage Current	−10		+10	μA	GND ≤ V_{OUT} ≤ V_{CC}
V_{IL}	Input Low Voltage	−0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = +1.6 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −400 μA
I_{CC}	V_{CC} Supply Current		85	120	mA	V_{CC} = 5V
P_{D1}	Operating Power Dissipation		425		mW	

CAPACITANCE T_A = 25°C, V_{CC} = GNDD = 0V

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
C_{IN}	Input Capacitance		5	10	pF	f_c = 1 MHz
$C_{I/O}$	Input/Output Capacitance		10	20	pF	
C_{OUT}	Output Capacitance		8	15	pF	Unmeasured pins returned to GNDD

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 5\%$, GNDD = 0V

SYSTEM CLOCK, SLD INTERFACE TIMING

Symbol	Parameter	Min	Max	Units
CLK	System Backplane Clock Frequency	1.536	4.096	MHz
	CLK Duty Cycle	45	55	%
	CLK Rise, Fall Times		10	ns
	Frame Synchronization Pulse Period	125		μs
t_{FS}	Frame Synchronization Pulse Width	60	t_{CLK}	ns
t_{dFS}	Pulse Delay to CLK	10		ns
t_{SFS}	Set-Up Time to CLK	50		ns
SCL	Slave Clock SCL Frequency	512	512	KHz
t_{dSCL}	SCL Delay Time from CLK	100	165	ns
SDIR	Slave Direction SDIR Frequency	8	8	KHz
t_{dDIR}	SDIR Delay Time to CLK	120	190	ns
t_{dSLD}	SLD Data Delay	160	300	ns
t_{DER}	Data Enable Receive	100	180	ns
t_{DDR}	Data Disable Receive	100	180	ns
t_{DEX}	Data Enable Transmit	0		ns
t_{DHX}	Data Hold Transmit	0		ns
t_{DSX}	Data Set-Up Transmit	$\frac{1}{2\,CLK} + 200$		ns
t_{DSIG}	Signaling Strobe Delay	110	160	ns

A.C. CHARACTERISTICS $T_A = 25°C$, V_{CC} = GNDD = 0V

MICROPROCESSOR INTERFACE

READ CYCLE

Symbol	Parameter	Min	Max	Units
t_{AL}	Address Set-Up to ALE	30		ns
t_{LA}	Address Hold after ALE	20		ns
t_{AA}	ALE Pulse Width	60		ns
t_{RD}	Data Delay from $\overline{RD}$		150	ns
t_{DF}	Data Float after $\overline{RD}$		25	ns
t_{RR}	$\overline{RD}$ Pulse Width	150	10^7	ns
t_{RI}	$\overline{RD}$ Control Interval(1)	$2 \times \frac{1}{CLK}$		ns
t_{RI}	$\overline{RD}$ Control Interval(2)	100		ns

WRITE CYCLE

Symbol	Parameter	Min	Max	Units
t_{DW}	Data Set-Up to $\overline{WR}$	50		ns
t_{WD}	Data Hold after $\overline{WR}$	25		ns
t_{WW}	$\overline{WR}$ Pulse Width	100		ns
t_{WI}	$\overline{WR}$ Control Interval(1)	$2 \times \frac{1}{CLK}$		ns
t_{WI}	$\overline{WR}$ Control Interval(2)	50		ns

NOTES:
1. Read or Write of BFF and XFF.
2. Read or Write of all other registers.

DMA READ

Symbol	Parameter	Min	Max	Units
t_{DMA}	DMA Read Time		$7 \times \frac{1}{CLK}$	ns
t_{DH}	DMOR Hold Time		75	ns
t_{AR}	Address Stable before $\overline{RD}$	0		ns
t_{RD}	Data Delay from $\overline{RD}$		150	ns
t_{DF}	Data Float after $\overline{RD}$		20	ns
t_{RA}	Address Hold after $\overline{RD}$	0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150	10^4	ns

DMA WRITE

Symbol	Parameter	Min	Max	Units
t_{DMA}	DMA Write Time		$7 \times \frac{1}{CLK}$	ns
t_{IH}	DMIR Hold Time		80	ns
t_{AW}	Address Stable before $\overline{WR}$	0		ns
t_{WA}	Address Hold after $\overline{WR}$	0		ns
t_{DW}	Data Set-Up to $\overline{WR}$	30		ns
t_{WD}	Data Hold after $\overline{WR}$	25		ns
t_{WW}	$\overline{WR}$ Pulse Width	100		ns

System Backplane Timing Parameters

PCM INTERFACE—RECEIVE TIMING

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{DSRR}	Receive Data Set-Up DCR = 0(1)	40		ns	60 ns for Interleaved Mode
t_{DHRR}	Receive Data Hold DCR = 0(1)	10		ns	
t_{DSRF}	Receive Data Set-Up DCR = 1(2)	20		ns	
t_{DHRF}	Receive Data Hold DCR = 1(2)	40		ns	

PCM INTERFACE—TRANSMIT TIMING

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{DZXR}	Data Enable DCX = 0	80	160	ns	C_L = 200 pF
t_{DHXR}	Data Hold Time DCX = 0	45	160	ns	C_L = 200 pF
t_{DZXF}	Data Enable DCX = 1	40	100	ns	C_L = 200 pF
t_{DHXF}	Data Hold Time DCX = 1	40	100	ns	C_L = 200 pF
t_{HZX}	Data Float after CLK TS	35	80	ns	C_L = 150 pF
t_{SONR}	Timeslot X to Enable DCX = 0	70	130	ns	C_L = 150 pF
t_{SONF}	Timeslot X to Enable DCX = 1	40	100	ns	C_L = 150 pF
t_{SOFF}	Timeslot X to Disable	40	100	ns	C_L = 150 pF

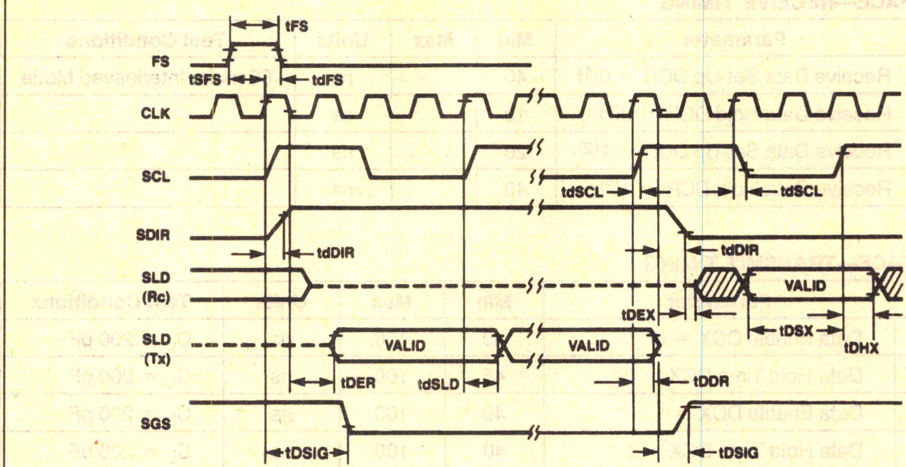
HDLC INTERFACE TIMING

Symbol	Parameter	Min	Max	Units	Test Conditions
t_{DS}	Receive Data Set Up	40		ns	
t_{DH}	Receive Data Hold	10		ns	
t_{TD}	Transmit Data Delay	40	100	ns	C_L = 200 pF
t_{HZX}	Data Float on TS Exit	35	80	ns	C_L = 200 pF
t_{SON}	Timeslot X to Enable	40	95	ns	C_L = 150 pF
t_{SOFF}	Timeslot X to Enable	35	90	ns	C_L = 150 pF

NOTES:
1. DCR = 0 data latched on rising edge of CLK.
2. DCR = 1 data latched on falling edge of CLK.

WAVEFORMS

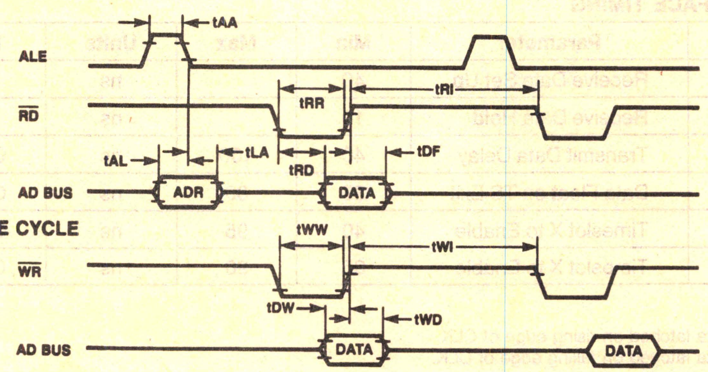
SYSTEM CLOCK, SLD INTERFACE TIMING



270155-5

Microprocessor Interface

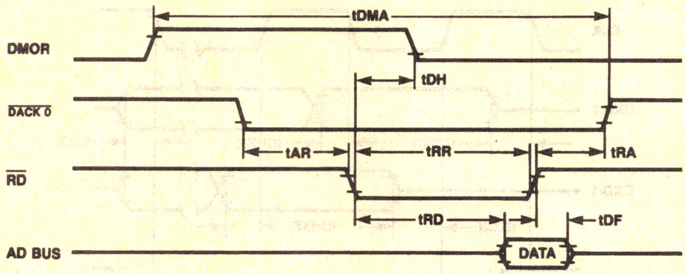
READ CYCLE



270155-6

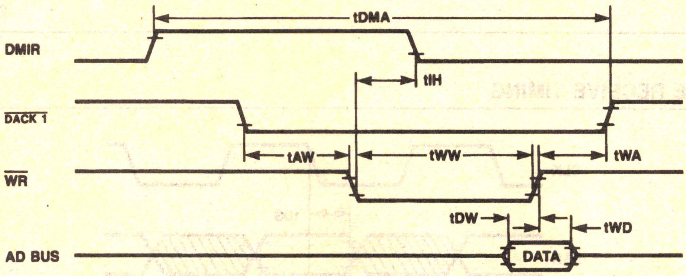
WAVEFORMS (Continued)

DMA READ



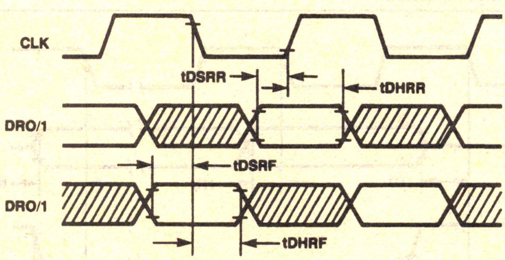
270155-7

DMA WRITE



270155-8

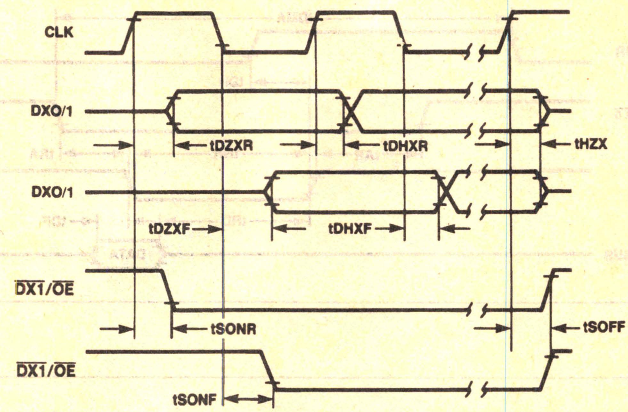
PCM INTERFACE RECEIVE TIMING



270155-9

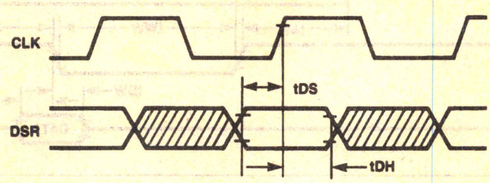
WAVEFORMS (Continued)

PCM INTERFACE TRANSMIT TIMING



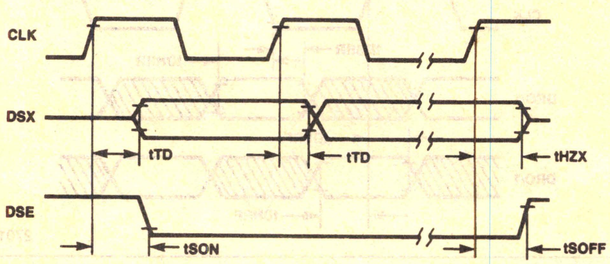
270155-10

HDLC INTERFACE RECEIVE TIMING



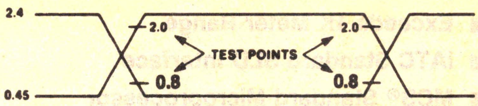
270155-11

HDLC INTERFACE TRANSMIT TIMING



270155-12

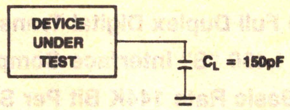
A.C. TESTING INPUT, OUTPUT WAVEFORM



270155–13

A.C. Testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

A.C. TESTING LOAD CIRCUIT



270155–14

C_L = 150 pF
C_L Includes Jig Capacitance

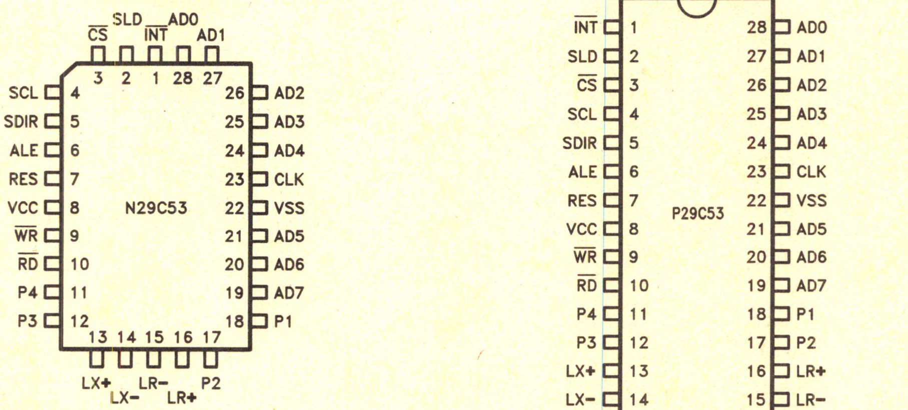
iATC 29C53 DIGITAL LOOP CONTROLLER

- 4-Wire Full Duplex Digital Transceiver
- CCITT I.430 "S" Interface Compatible
- ISDN Basic Rate 144K Bit Per Second
- D-Channel Processing Support
- Point-To-Point or Point-To-Multipoint Bus Configuration
- Same Device Used at Both Ends of Loop
- Exceeds 1K Meter Range
- iATC Standard SLD Interface
- MCS® Standard Microprocessor Interface
- Peripheral Interface/Status Port
- Low Power, High Density CHMOS
- Single +5 Volt Supply

The Intel Advanced Telecommunication Component (iATC) 29C53 Digital Loop Controller (DLC) is a 4-wire transceiver/controller that is CCITT I.430 compatible and can function at either loop end. This part has integrated those features which are pertinent to the transceiver function; yet it offers efficient interfacing to other system components such as combos, line card controllers and MCS® microcontrollers through the SLD and microprocessor interface ports. It is primarily intended for use in Integrated Services Digital Networks (ISDN) as a basic rate digital data transceiver which transfers data at 144 Kbps as three separate channels—two 64 Kbps digitized-voice/data channels (B-channels), and a 16 Kbps signaling/data channel (D-channel). The B- and D-channel routing along with D-channel processing (packetization) is programmable through either the microprocessor or SLD interface ports. The 29C53's loop interface uses a 100% pulse-width pseudo-ternary inverted line code similar to Alternate Mark Inversion, which meets CCITT's "S" interface recommendations. It is capable of interfacing with up to eight 29C53s in a passive or extended bus configuration as well as point-to-point.

Plastic Chip Carrier*
350 x 550 MILS

28-Lead Dual-In-Line Package



270097–1

270097–2

*Call sales representative for availability

Figure 1. 29C53 Pin Configurations

November 1986
Order Number: 270097-002

Symbol	Pin No.	Function
V_{CC}	8	**POSITIVE SUPPLY:** Input voltage is +5V ±5%.
V_{SS}	22	**GROUND:** 0V
CLK	23	**MASTER CLOCK:** The 3.84 MHz system clock input is the reference for the loop and the SLD interface.
Res	7	**RESET:** (Active high input). A high level on this pin initializes most control registers and places most interface outputs in a high impedance state. Operation begins when the high level is removed.
LX+, LX−	13, 14	**POLARIZED TRANSMIT LOOP INTERFACE PINS:** These pins will directly drive the twisted pair line through a 2.5:1 line transformer. The transmitted line code is similar to alternate mark inversion.
LR−, LR+	15, 16	**RECEIVE LOOP INTERFACE PINS:** The receiver is not sensitive to polarity.
SLD	2	**SUBSCRIBER LINE DATALINK:** This pin transfers serial data between the 29C53 and other SLD based components (e.g., 29C51, 2952, 29C55, etc.).
SCL	4	**SUBSCRIBER CLOCK:** 512 KHz signal may be either generated or received by the 29C53. This signal clocks the data on the SLD pin.
SDIR	5	**SUBSCRIBER DIRECTION:** An 8 KHz signal may be either generated or received by the 29C53 to indicate SLD data direction and framing. A high level indicates and enables master to slave transfer; a low level indicates slave to master transfers.
$\overline{CS}$	3	**CHIP SELECT:** (Active low input). A low level on this pin enables the 29C53 bus interface for the next bus cycle. The value is latched by the falling edge of ALE.
$\overline{RD}$	10	**READ STROBE:** (Active low input). When low, data is transferred from the selected register to the data pins AD (0–7). When no local microprocessor is connected, this pin should be tied to V_{SS}.
$\overline{WR}$	9	**WRITE STROBE:** (Active low input). When $\overline{WR}$ changes from low to high, data on pins AD (0–7) is latched into the 29C53. When no local microprocessor is connected, this pin should be tied to V_{SS}.
AD (0–7)	19–21, 24–28	**ADDRESS/DATA PINS:** This is a standard MCS microprocessor bus used to transfer address and data between the local microprocessor and the internal registers of the 29C53. When a local microprocessor is not used, these pins should be tied to V_{SS}.
ALE	6	**ADDRESS LATCH ENABLE:** Address is latched from AD(0–7) on falling edge of this signal. State of $\overline{CS}$ is also latched at this time.
$\overline{INT}$	1	**INTERRUPT REQUEST:** This is an open drain active low output. (See text for the interrupt conditions.)
P1, P2	18, 17	**PERIPHERAL INTERFACE INPUTS:** These are standard CHMOS high impedance inputs that are sampled at a 4 KHz rate (once per "s" frame). The sampled data is stored in the LPS register (bits 5 and 6). If any peripheral input bits have changed value since the previous frame, an interrupt condition is indicated; only present status is available.
P3	12	**PERIPHERAL INTERFACE INPUT/OUTPUT PIN:** When configured as an input, this pin has the same characteristics as P1 and P2. The sampled data is stored in the LPS register (bit 7). When programmed as an output, this pin outputs the data stored in the PEC register (bit 1). The pin is configured by bit 2 of the PEC register. An alternate function of this pin and P4 is to indicate the status of the SLD interface. See the section on the SLD interface.
P4	11	**PERIPHERAL INTERFACE OUTPUT PIN:** This pin outputs data stored in the PEC register (bit 0) or SLD status.

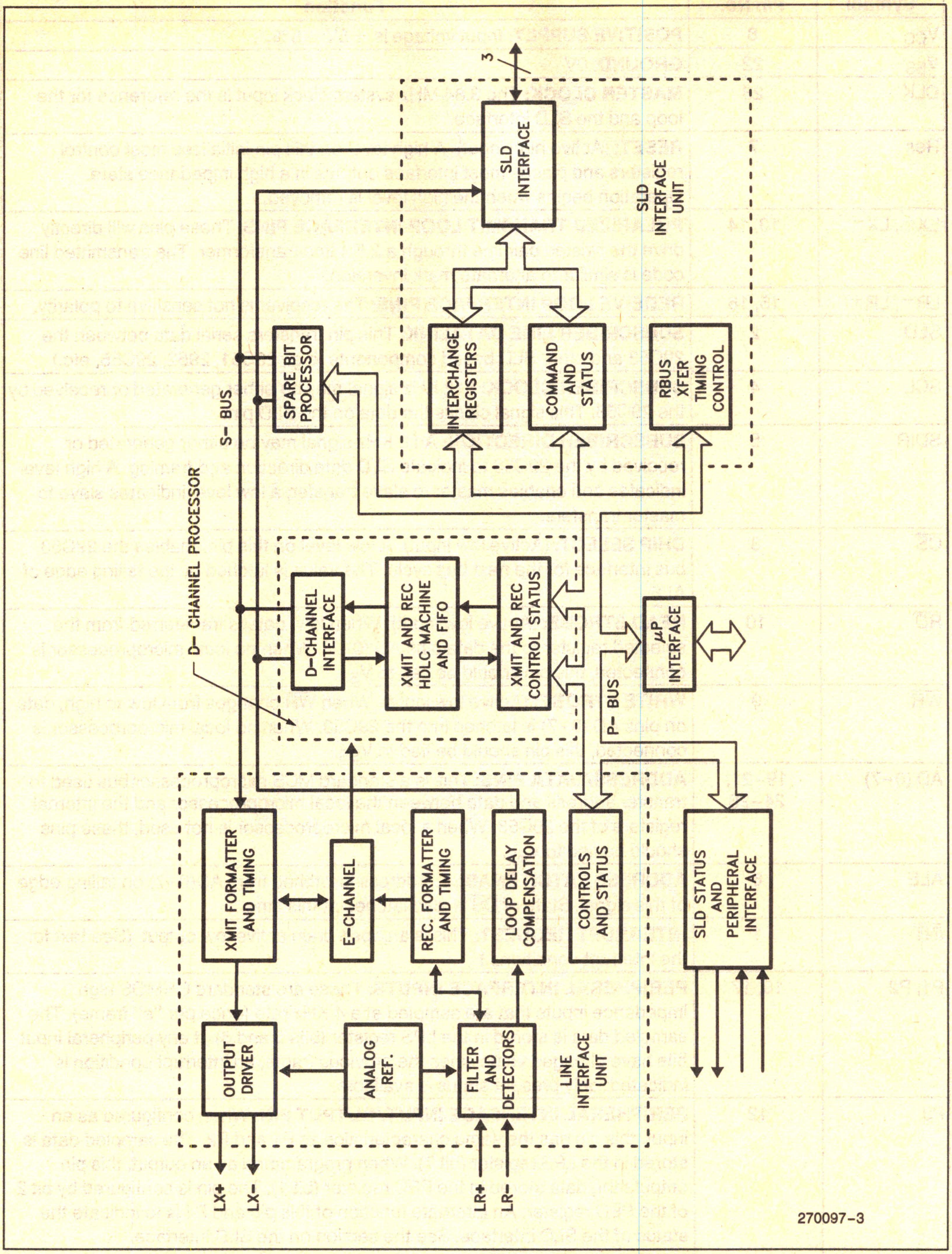


Figure 2. 29C53 Block Diagram

29C53 FUNCTIONAL DESCRIPTION

The 29C53 Digital Loop Controller is a multi-channel ISDN transceiver which provides a multiplicity of advanced communication functions and services. The 29C53 allows the extension of digital voice and data directly to subscriber equipment over a 192 Kbps baseband 4-wire serial interface. The 29C53 is a 28-pin device which, when used with the iATC 29C51, 2952 and a microprocessor implements flexible point-to-point or point-to-multipoint CCITT I.430 compatible voice/data communications. It is primarily intended for use in PBX's and ISDN terminal equipment.

The 29C53 may be incorporated at either end of a subscriber loop interface (at the line card or digital telephone/terminal). As shown in Figure 2, the 29C53 has four separate interfaces: a serial SLD iATC Telecom system interface; a parallel peripheral interface; a parallel microprocessor interface and a 4-wire CCITT compatible S-interface (subscriber loop interface).

THE BLOCK DIAGRAM

Figure 2 represents a block diagram of the 29C53. Its three major blocks, the line interface unit, the D-channel processor and the SLD interface unit are interconnected by two buses. The parallel bus (P-bus) is used to transfer processed D-channel data and general status and control information, while the serial bus (S-bus) is used to transfer B-channel data and unprocessed D-channel data between the line interface unit and the SLD interface unit.

The SLD interface unit consists of shift registers and serial to parallel converters. Data from both the S-bus and the SLD interface is stored here in appropriate parallel registers before it is loaded into shift registers and passed on. All of the timing circuitry for the SLD interface is located here. This block also contains a command processor which is responsible for controlling the functionality of the chip.

The D-channel processor has three major sections. An HDLC section performs some of the basic LAPD protocol functions such as zero insertion or deletion, flag recognition or insertion for frame delineation, abort flag recognition, idle state transmission, and end of packet frame check sequence for both data directions. The FIFO section consists of two 32-byte buffers, one for transmit and one for receive. The control and status section monitors the FIFO data levels and the HDLC section for progress. Interrupts or requests for service may be generated for conditions such as a full or empty FIFO, loss of sync, frame check error, overflows and aborts.

The line interface unit contains the line drivers and receivers for the S interface. Connection is made to the transmission lines through a 2.5:1 line transformer. Formatting, timing and synchronization are also provided here. The receiver includes filters, AGC circuitry, threshold detectors and a loop delay shift register. The loop delay shift register maintains the proper internal frame relationship regardless of loop length (it allows extra propagation delay time for long loops or line repeaters). The received D-channel bits are logically looped back to create the E-channel bits in an NT application through the E-channel circuitry.

The microprocessor interface circuitry allows the 29C53 to function as a peripheral to an MCS microcontroller. The internal P-bus actually becomes an extension of the microcontroller's bus. All internal registers are directly accessible.

The spare bits processing block provides access to all the miscellaneous bits in the S frame except for the framing bits and the balance bits. When spare bit functions become defined, they can be easily monitored and modified.

The peripheral interface circuitry provides an auxiliary port for controlling auxiliary peripherals such as power controllers, etc.

SLD INTERFACE

The SLD interface provides half-duplex 512 Kbps communication with other devices incorporating SLD interfaces (such as line card controllers and codec/filters). Of the 256 Kbps in each direction, 128 Kbps is dedicated to voice/data channels B1 and B2. The remaining bandwidth is used for the D-channel data and various control and status transfers depending on the exact application.

As shown in Figure 3, the SLD interface consists of three lines: the SLD bidirectional data line; the 512 KHz SCL clock line; and the 8 KHz SDIR data direction line. SLD data is updated on the rising edge of SCL and is latched on its falling edge. The 125 μs SLD frame period consists of 32 bits transferred in master to slave direction followed by 32 bits in the slave to master direction. The 32 bits compose four 8-bit bytes in the following order: B1 and B2 (voice or data bytes); C (control byte); and S (signaling or status byte). Unprocessed D-channel data may be transported over the S-byte in bits 0 and 1, or over the B2 byte.

The 29C53 can be operated as an SLD master or slave. As an SLD master, it generates the SCL and SDIR signals. When SDIR is high, the SLD pin outputs data. As a slave, it receives SCL and SDIR sig-

nals and SDIR enables the SLD output driver when it is low. The SLD bus is always active; no powered-down or inactive mode is defined.

In a network termination (NT) application (line card), whether a microprocessor is connected to the 29C53 or not, the SLD control and signal bytes may be used for 29C53 configuration and D-channel transfers. The command bytes are interpreted and executed by the 29C53's command processor circuit. The command processor generates internal P-bus cycles to carry out those commands. Internal prioritization resolves P-bus collisions between microprocessor-interface generated and command-processor generated cycles. In case of collisions, the microprocessor interface has higher priority to minimize access time but both cycles will be completed.

"S" TRANSCEIVER

The 4-wire "S" transceiver circuit in the 29C53 conforms to CCITT recommendation. I.430. This transceiver provides the internal drivers for transformer coupling to standard telephone type twisted pair cables.

The "S" transceiver line code is 100% Pseudo-Ternary Inverted code which is similar to Alternate Mark Inversion, except that logical ones are transmitted as spaces, logical zeros as marks. A space has a nominal differential voltage of zero volts. Marks may be either positive or negative differential signals, nominally 0.75 volts in amplitude. Marks alternate polarity except when a "code violation" is created for establishing frame reference timing.

The nominal bit rate is 192 Kbps. Figure 4 shows the frame structure. The 250 μs frame transfers two octets of B1, B2 and four bits of D data. The E bits in the master to slave direction, echo received D-channel data. The "S" interface slave compares the receive E-channel data to its transmitted D-channel data for D-channel contention as defined in CCITT recommendation I.440. If these bits do not agree, then the slave will abort its transmission effort. The S, FA, and N bits are all accessible and programmable.

The activation protocol described in I.430 is supported by the 29C53. An inactive receiver can achieve bit synchronization to an incoming signal with approximately 30 mark-mark transitions. Info 2 or 3 frame alignment is not officially recognized until reception of 16 frames, to allow settling of the 29C53's adaptive receive data thresholds. The full activation sequence will complete in approximately 10 ms.

The 29C53 is not sensitive to the polarity of the wire pair connected to LR+ and LR−. Marks are always interpreted as zeros and framing relies on violations; not on absolute polarity. System configurations may dictate that care be taken in connecting the LX outputs. In a multi-drop bus configuration all TE transmitters must be connected with the same polarity so that positive mark to negative mark contention does not take place in the framing and D-channel bits.

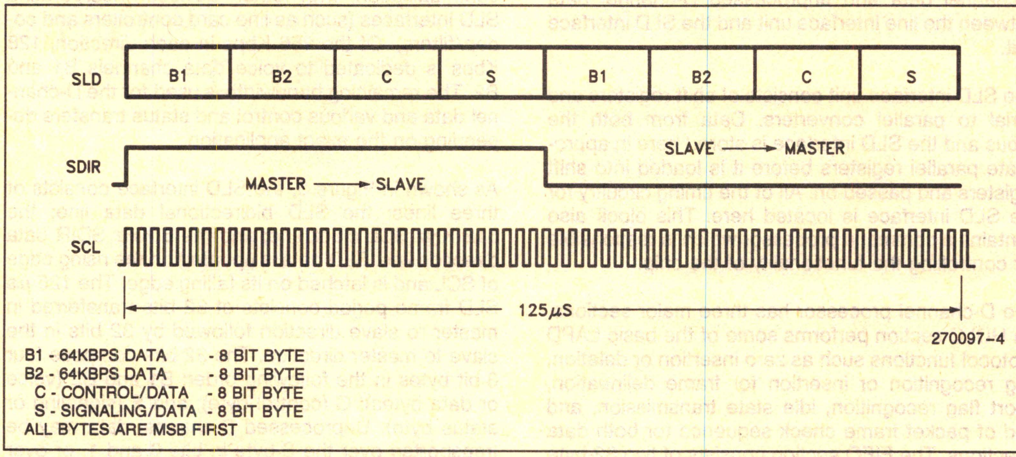
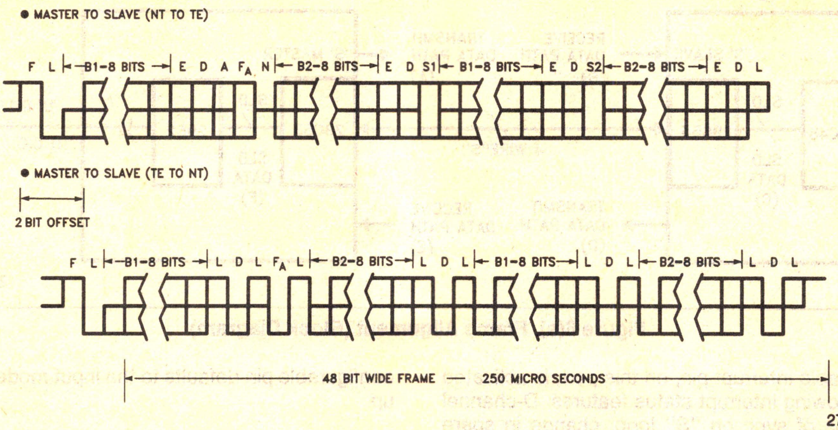


Figure 3. SLD Interface



A - BIT USED FOR ACTIVATION
B1, 2 - 64 K BPS DATA
D - DCHANNEL BIT (16K BPS DATA)
E - D CHANNEL ECHO BIT
F - FRAME BIT

F_A - AUX. FRAME BIT
L - DC BALANCING BIT
N - BIT = $\overline{F_A}$
S1, 2 - RESERVED FOR FUTURE STANDARDIZATION

Figure 4. The S-Interface Frame Structure

The 29C53, functioning as an "S" interface master in a multi-drop application, can interface with up to eight slave systems. In this multiplexing operation a slave initiates a data transfer to the master, by requesting access and transferring the data in accordance with the D-channel line access protocol (I.440). Figure 5 shows typical applications of the 29C53.

The frame alignment timing diagram Figure 6(b) shows the relationship of the "S" interface data to the SLD data. Figure 6(a) shows the block diagram used for the timing diagram. The top timing diagram shows the transmitted "S" data stream from the network terminator (master). The dotted lines depict up to 20 μs propagation delay to the "S" receiver at the terminal equipment (slave) end. The terminal equipment's transmitted "S" interface frame is designed to have a fixed 2-bit frame alignment delay from that of its received frame. The adjustment for loop propagation delay is accounted for in the network terminator's receive circuitry (loop delay section of block diagram). The loop delay circuitry will compensate for up to 10 bit periods of round trip propagation delay which allows line repeaters to be placed in a loop that is several thousand meters long.

MICROPROCESSOR INTERFACE

This interface is designed to operate with standard Intel microprocessors such as the MCS®-48, MCS-51, MCS-85 and iAPX-86 families. All of the 29C53's internal registers are accessible and most are available by a single microprocessor cycle access.

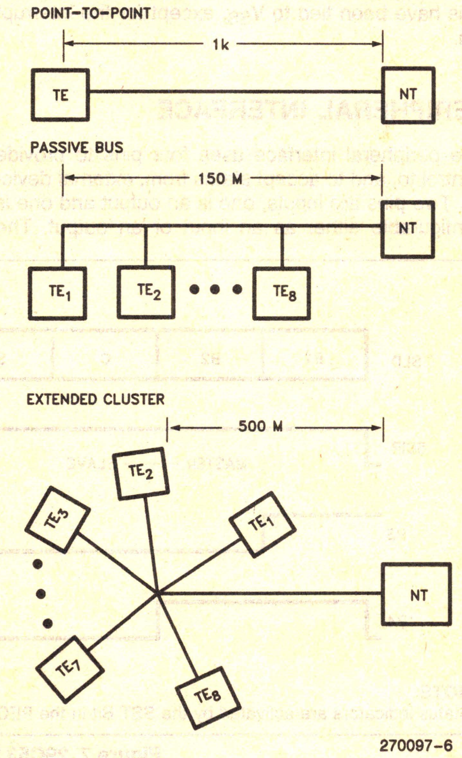


Figure 5. 29C53 Bus Configurations

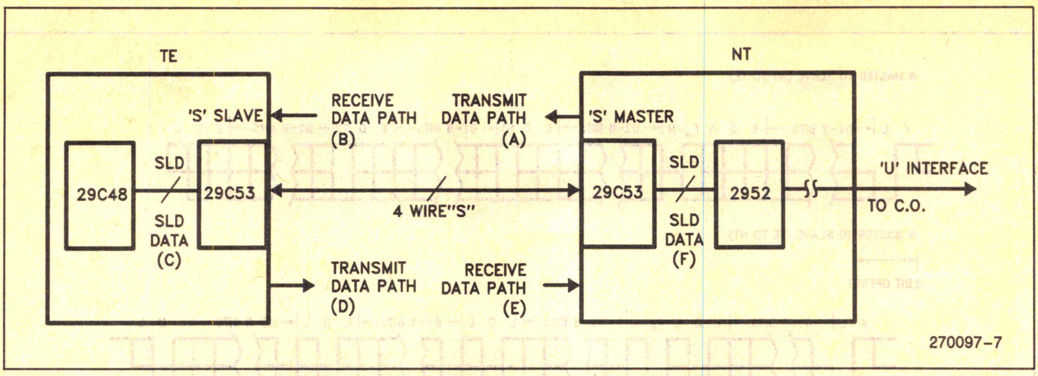


Figure 6(a). Frame Alignment (Block Diagram)

The maskable interrupt pin, on this port, is activated by the following interrupt status features: D-channel errors; loss of sync on "S" loop; change in spare bits or peripheral interface data; FIFO data transfer requests.

Alternatively, the 29C53 can operate in the stand-alone mode in line card and NT applications. This mode is determined on a power-up condition or after a reset, provided all the microprocessor interface pins have been tied to V_{SS}, except for the interrupt pin.

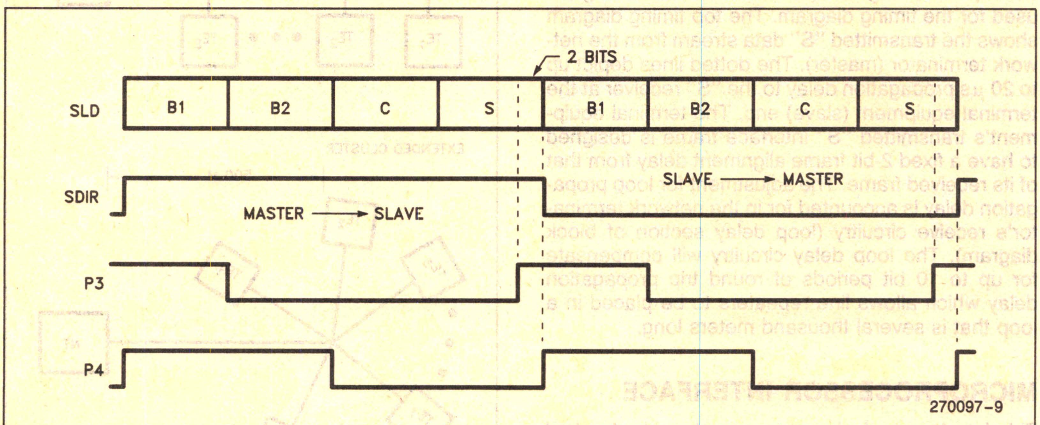
PERIPHERAL INTERFACE

The peripheral interface uses four pins to provide control to, and to accept status from, external devices. Two pins are inputs, one is an output and one is configurable either as an input or an output. The configurable pin defaults to the input mode on power up.

The peripheral interface can also be used to indicate SLD status. Figure 7 shows the timing diagram of P3 and P4. B1, B2 and D-channel data on the SLD pin can be selected or gated by using these signals. As noted on the P3 timing, the D-channel is imbedded in the last two bits (0, 1) of the signaling byte.

INTERNAL CONTROL AND STATUS REGISTERS

All of the 29C53's internal control and status registers may be accessed through the microprocessor interface or through the SLD interface. When a microprocessor accesses a register, the address and CS inputs are latched on the trailing edge of ALE.



NOTE:
Status indicators are activated by the SST Bit in the PEC Register

Figure 7. 29C53 SLD Status Indicators

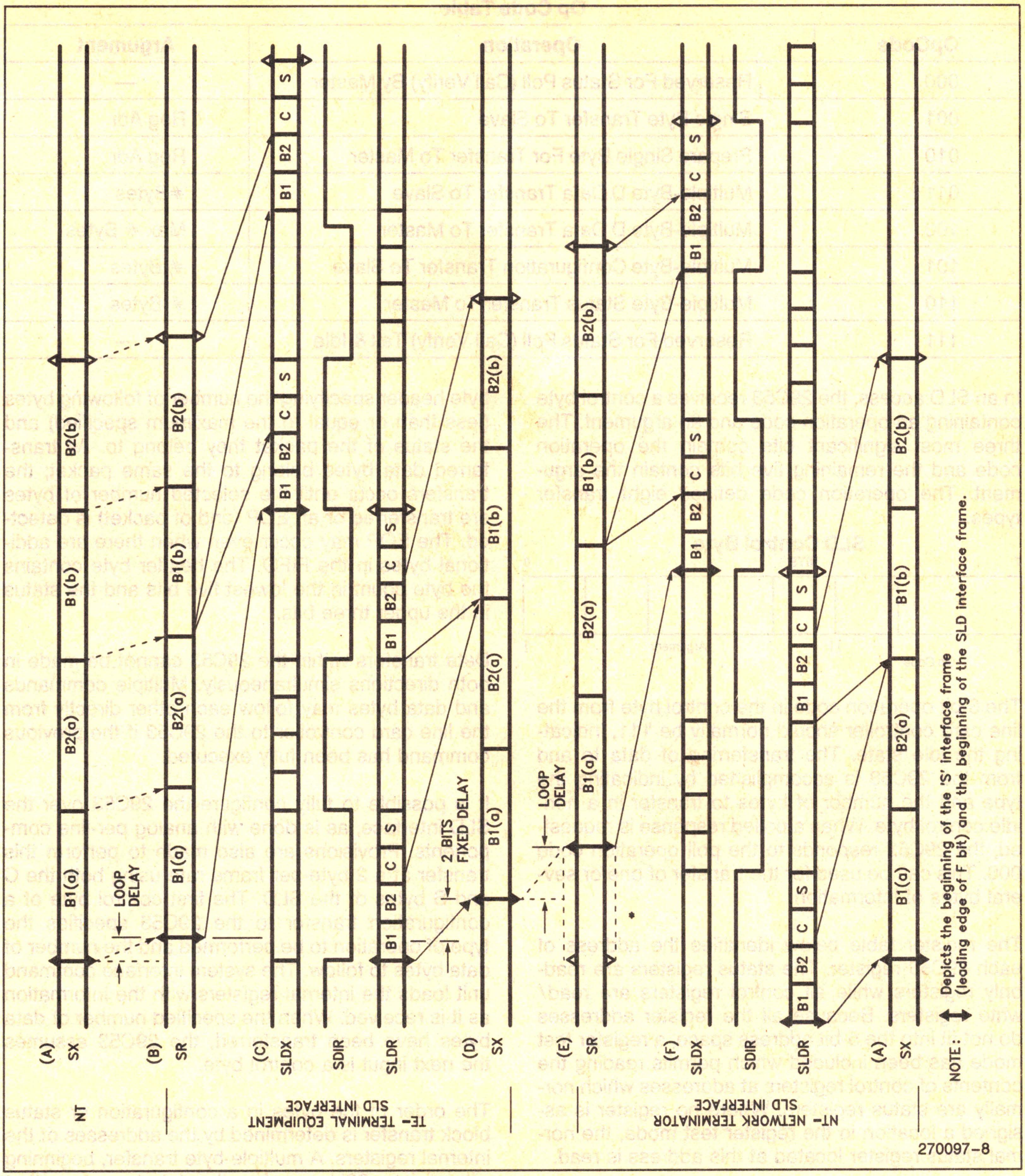


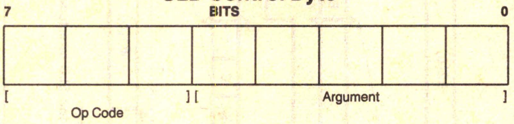
Figure 6(b). Frame Alignment (Timing Diagram)

Op Code Table

OpCode	Operation	Argument
000	Reserved For Status Poll (Call Verify) By Master	—
001	Single Byte Transfer To Slave	Reg Adr
010	Prepare Single Byte For Transfer To Master	Reg Adr
011	Multiple-Byte D Data Transfer To Slave	#Bytes
100	Multiple-Byte D Data Transfer To Master	Max # Bytes
101	Multiple-Byte Configuration Transfer To Slave	# Bytes
110	Multiple-Byte Status Transfer To Master	# Bytes
111	Reserved For Status Poll (Call Verify) Tail & Idle	—

In an SLD access, the 29C53 receives a control byte containing an operation code and an argument. The three most significant bits contain the operation code and the remaining five bits contain the argument. The operation code defines eight transfer types.

SLD Control Byte



The 3-bit operation code in the control byte from the line card controller should normally be 111, indicating the idle state. The transferring of data to and from the 29C53 is accomplished by indicating the type and the number of bytes to transfer in a non-idle control byte. When a polled response is requested, the 29C53 responds to the poll operation code 000. This can be used for the transfer of one or several bytes of information.

The register table below identifies the address of each 29C53 register. The status registers are read-only registers while all control registers are read/write registers. Because all the register addresses do not fit into the 5-bit address space, a register test mode has been included which permits reading the contents of control registers at addresses which normally are status registers. Where no register is assigned a location in the register test mode, the normal status register located at this address is read.

The D-channel block transfers from the 29C53 to the line card controller preface the data bytes with a byte header specifying the number of following bytes (less than or equal to the maximum specified) and the status of the packet they belong to. All transferred data bytes belong to the same packet; the transfers occur until the selected number of bytes are transferred or an EOP (end of packet) is detected. The EOP may occur even when there are additional bytes in the FIFO. The header byte contains the byte count in the lowest five bits and the status in the upper three bits.

Data transfers within the 29C53 cannot be made in both directions simultaneously. Multiple commands and data bytes may follow each other directly from the line card controller to the 29C53 if the previous command has been fully executed.

It is possible to fully configure the 29C53 over the SLD interface, as is done with analog per-line components. Provisions are also made to perform this transfer at a 2 byte-per frame rate using both the C and S bytes of the SLD. The first control byte of a configuration transfer to the 29C53 specifies the type of operation to be performed and the number of data bytes to follow. The system interface command unit loads the internal registers with the information as it is received. When the specified number of data bytes have been transferred, the 29C53 assumes the next input is a control byte.

The order of the bytes in a configuration or status block transfer is determined by the addresses of the internal registers. A multiple-byte transfer, beginning with register 00H, transfers the data to or from that register and increments the address counter.

Table 1. 29C53 Registers

Address(1)	Access	Symbol	Name
00000	RD	IXS	Interrupt Status
00000	WR (RT)	IMR	Interrupt Mask
00001	RD	DPS	D-Channel Processor Status
00001	WR (RT)	DPC	D-Channel Processor Control
00010	RD	LPS	Loop and Peripheral Interface Status
00010	WR (RT)	LCR	Loop Interface Control
00011	RDWR	PEC	Peripheral Interface and E-Channel Control
00100	RD	RFN	Receive FIFO Status - # of Bytes Used
00100	WR (RT)	SCR	SLD Interface Control
00101	RD	XFN	Transmit FIFO Status - # of Free Bytes
00101	WR (RT)	SDC	SLD Data Transfer Configuration
00110	RD	SBR	Spare Bits Receive Status
00110	WR (RT)	SBX	Spare Bits Transmit
00111	RDWR	LLB	Loop Interface Loopback Control
01000	RD	RFO	Receive FIFO Output
01001	WR	XFI	Transmit FIFO Input
01010	RDWR	GCR	General Command Register
01011	RDWR	DPR	D-Channel Priority Counter
01100	RDW	RFIL	Receive FIFO Interrupt Level
01101	RDWR	XFIL	Transmit FIFO Interrupt Level
01110	RD	PLENH	Packet Length High Byte
01110	WR (RT)	DUTH	D-Channel Byte Counter Underflow and Overflow Threshold
01111	RD	PLENL	Packet Length Low Byte
01111	WR (RT)	DOTH	D-Channel Byte Counter Overflow Threshold
10000	RDWR	AFD	Auxiliary Frame/Multiframe Division
10001	RDWR	PSR	Position Selection
10010	RD	RSR	Receive Service Request
10011	RD	XSR	Transmit Service Request
11000	RDWR	B1LS	B1 Data in Loop to SLD Direction
11001	RDWR	B2LS	B2 Data in Loop to SLD Direction
11010	RDWR	CR	Control Byte from SLD
11011	RDWR	SR	Signaling Byte from SLD
11100	RDWR	B1SL	B1 Data in SLD to Loop Direction
11101	RDWR	B2SL	B2 Data in SLD to Loop Direction
11110	RDWR	CX	Control Byte to SLD
11111	RDWR	SX	Signaling Byte to SLD

NOTE:
1. Address represents AD1–AD5. AD0 is not used by the 29C53 for addressing.

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
Voltage on any Pin .... V_{SS} −0.5V to V_{CC} to +0.5V
Maximum Voltage on V_{CC}
with Respect to V_{SS} .... +7V
Total Power Dissipation .... 500 mW

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS V_{CC} = +5V ±5%; V_{SS} = 0V; T_A = 0°C to 70°C;

Typical Values are at T_A = 25°C and Nominal Power Supply Values

DIGITAL INTERFACES

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current (Excluding LR+, LR−)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
V_{IL}	Input Low Voltage	−0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = +2.0 mA
V_{OH1}	Output High Voltage	2.4			V	I_{OH} = −400 μA
V_{OH2}	Output High Voltage	0.9 V_{CC}			V	I_{OH} = −40 μA

POWER SUPPLY CURRENT (Averaged over 1 ms)

Symbol	Parameter	Min	Typ	Max	Units	Comments
I_{CC} (P)	Power Down (Standby)		4		mA	SLD and CLK Active
I_{CC} (I)	Idle Operating Current		8		mA	Receiver, SLD, OSC Active
I_{CC} (N)	Normal Operating Current		20		mA	Everything is Active (Excluding Current for Output Loads)

A.C. Characteristics V_{CC} = 5V ±5%; V_{SS} = 0V; T_A = 0°C − 70°C; CLK = 3.84 MHz

RECEIVER

Symbol	Parameter	Min	Typ	Max	Units	Comments
V_{RD}	Received Differential Mark Voltage	200		3000	mV	
Z_{IR}	LR+, LR− Input Impedance		60		kΩ	Each Pin
C_{IR}	LR+, LR− Input Capacitance		30		pF	Each Pin

TRANSMITTER

Symbol	Parameter	Min	Typ	Max	Units	Comments
V_{XD}	Transmit Differential Mark Voltage	1780		1980	mV	$200\ \Omega < R_L < 2.5\ k\Omega$
Z_{OX}	LX+, LX− Output Impedance		60		kΩ	Each Pin
C_{OX}	Output Capacitance		30		pF	Each Pin
R_L	Resistive Load Between LX+, LX−	200			Ω	
C_L	Capacitive Load Between LX+, LX−			1500	pF	
t_{LD}	Load Time Constant			0.5	μs	$R_L = 300\Omega$, $C_L = 1500$ pF
t_{MR}	Transmit Mark Rise Time			400	ns	Note 1
t_D	Damping Time Constant			1.5	μs	
I_{XL}	Source, Sink Current Limit		18		mA	
V_{XL}	Voltage Limiting		125		%	Nominal Mark Voltage

TIMING

Symbol	Parameter	Min	Typ	Max	Units	Comments
J	Timing Extraction Jitter ("S" Slave Mode)	−5		+5	%	I.430 8.2.2
PD	Total Phase Deviation LX with Respect to LR	−7		+15	%	I.430 8.2.3

NOTE:
1. Risetime is measured as 10% to 90% for space mark transitions and 90% to 0% and 0% to 90% for mark-to-mark transitions with respect to final value.

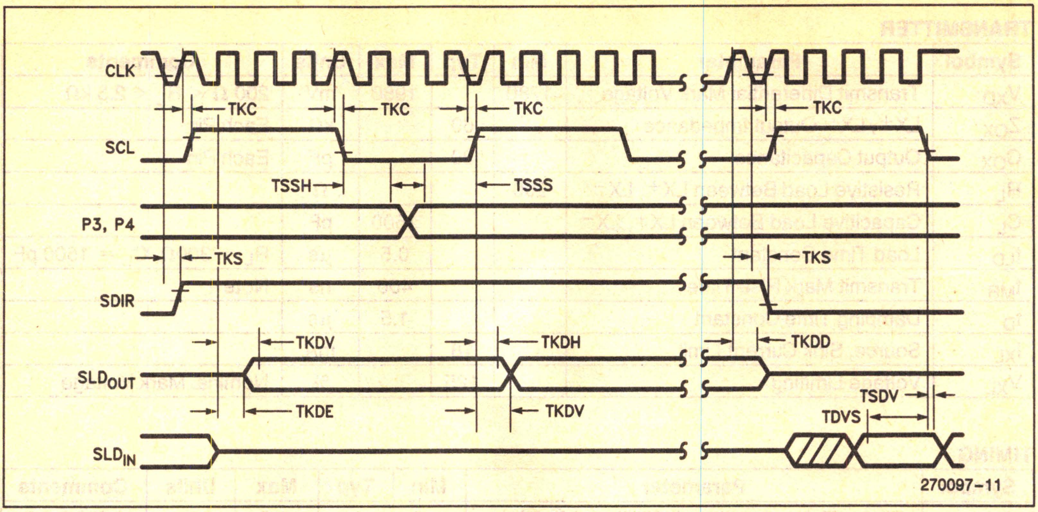


Figure 8. SLD Interface Timing (29C53 As Master)

SLD INTERFACE TIMING (29C53 as Master)

Symbol	Parameter	Min	Max	Units
TKC	CLK to SCL Delay		150	ns
TKS	CLK to SDIR Delay		150	ns
TKDE	CLK to SLD Driver Enabled	0		ns
TKDV	CLK to SLD Data Valid		150	ns
TKDH	SLD Data Hold After Clock Edge	0		
TKDD	CLK to SLD FLoat		150	ns
TDVS	SLD Data Input Setup Time to SCL	50		ns
TSDV	SLD Data Hold Time After SCL	80		ns
TSSH	SLD Status Hold After SCL	80		ns
TSSS	SLD Status Setup Before SCL	80		ns

NOTES:
1. 29C53 samples SLD input data on SCL falling edge.
2. 29C53 changes SLD output data on SCL rising edge.
3. 29C53 SLD out is enabled one CLK cycle after the SDIR rising edge and disabled one CLK cycle before the SDIR falling edge (as master only).

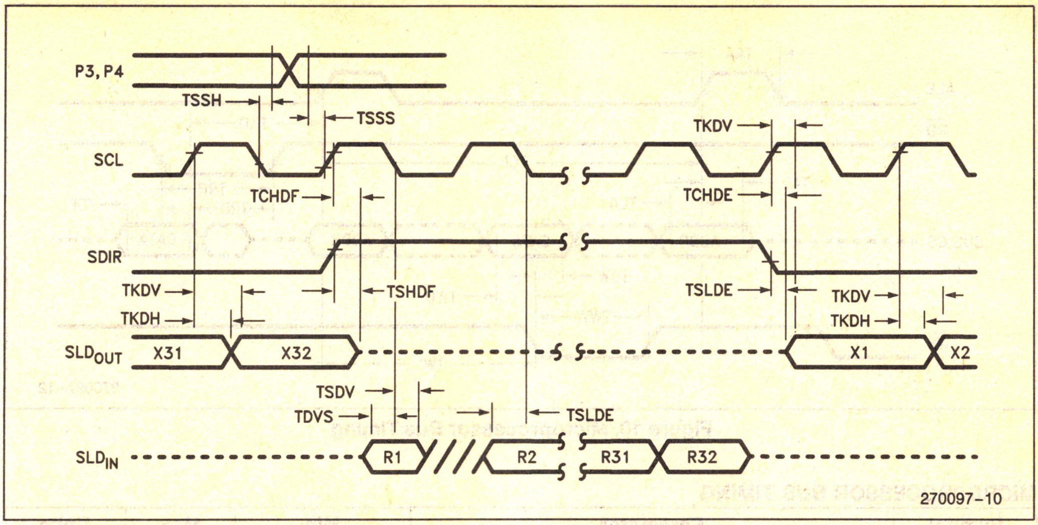


Figure 9. SLD Interface Timing (29C53 as Slave)

SLD INTERFACE TIMING (29C53 as Slave)

Symbol	Parameter	Min	Max	Units
TCHDF	SCL High to Data Out Float		50	ns
TSHDF	SDIR High to Data Out Float		50	ns
TKDH	Output Data Hold After SCL Edge	0		
TKDV	Output Data Valid After SCL Edge		100	ns
TDVS	SLD Input Data Setup Time	50		ns
TSDV	SLD Input Data Hold Time	80		ns
TCHDE	Enable SLD Output After SCL	0		ns
TSLDE	Enable SLD Output After SDIR	0		ns
TSSH	SLD Status Hold After SCL	80		ns
TSSS	SLD Status Setup Before SCL	80		ns

NOTES:
1. 29C53 samples SLD input data on SCL falling edge.
2. 29C53 changes SLD output data on SCL rising edge.

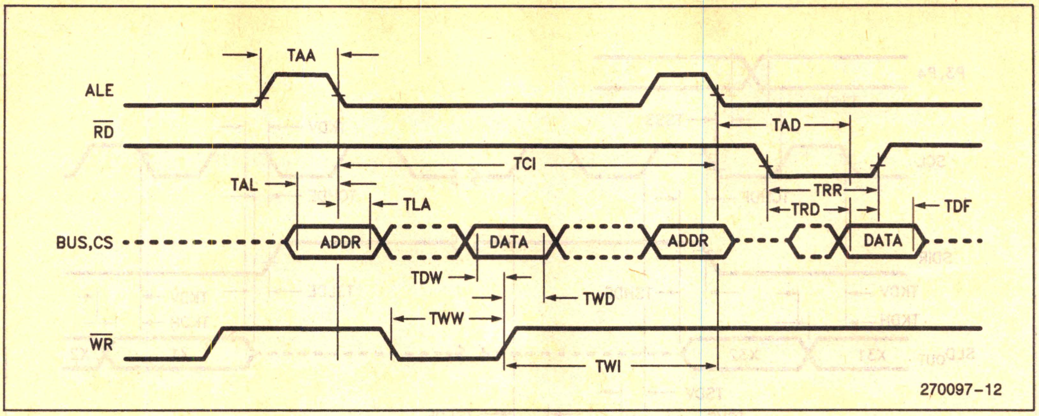


Figure 10. Microprocessor Bus Timing

MICROPROCESSOR BUS TIMING

Symbol	Parameter	Min	Max	Units
TAL	Address Setup Before ALE Trailing Edge	30		ns
TLA	Address Hold After ALE Trailing Edge	20		ns
TWW	Write Control Signal Width	100		ns
TDW	Data Setup Before $\overline{WR}$ Trailing Edge	40		ns
TWD	Data Hold After $\overline{WR}$ Trailing Edge	20		ns
TAA	ALE Pulse Width	60		ns
TWI	Active $\overline{CS}$ Cycle Disallowed After $\overline{WR}$(1)	2 × 1/CLK		
TRR	Read Control Signal Width	100		ns
TRD	Access Time from $\overline{RD}$ Leading Edge		80	ns
TAD	Access Time from ALE Trailing Edge		260	ns
TDF	Float Delay After $\overline{RD}$ Trailing Edge		40	ns
TCI	Active $\overline{CS}$ Cycle Time for FIFO Access, RFO XFI	3 × 1/CLK 4 × 1/CLK		

NOTE:
1. Allow 3 extra clock cycles for GCR commands to execute.

LCDK—29C51/52
iATC-29C51/52 LINE CARD DEVELOPMENT KIT

- Single Board Line Card Plus Group Controller
- Supports iATC 29C48, 29C50, and 29C51 Combo Chips
- Minimal Assembling, Low Cost, Kit Form
- Extensive System Software in ROM
- Wire-Wrap Area for Custom Circuitry
- A Comprehensive User's Manual
- Direct Interface with a CRT Terminal
- User's Manual

The LCDK-29C51/52 contains most of the components required for a full feature line card. The kit consists of a primary and a secondary board, which with an addition of few components can be made to function as a PBX. Included is a preprogrammed ROM containing a system monitor for general software utilities and system diagnostics. The SLD LCDK-29C51/52 is designed to be operated with a dumb terminal; however, software is provided to operate the kit from an Intellec Development System. This kit is an inexpensive and highly flexible prototype system that has been designed to reduce system development time thereby leading to an increased productivity.

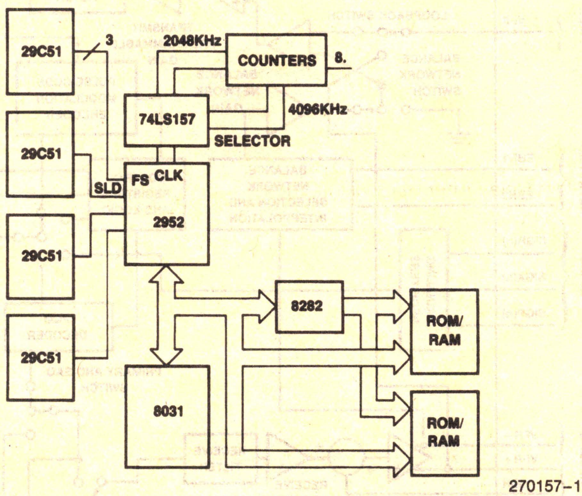


Figure 1. LCDK-29C51/52

August 1986
Order Number: 270157-001

FUNCTIONAL DESCRIPTION

The LCDK-29C51/52 consists of two boards, namely a primary and a secondary. The kit supports up to 128 TDM timeslots and with a little additional hardware, a full feature line card. The kit as supplied, without any additional components, can be used to form a signal path between two analog sources. The LCDK comes completely assembled. The LCDK-29C51/52 secondary board functional block diagram is shown in Figure 1, the primary board is a subset of the secondary.

29C51 Combo Family

The Intel iATC 29C51 Feature Control Combo is an advanced user-programmable, fully integrated PCM codec with transmit/receive filters fabricated in CMOS technology. The 29C51 realizes the same excellent transmission performance as the Intel 2913/2914 Combo while achieving the low power consumption typical of CMOS circuits.

The 29C51 incorporates the Supervision, Coding, Hybrid and Testing Operations out of the normal BORSCHT functions associated with an analog line interface circuit. The 29C51 also offers a secondary analog channel, programmable transmit and receive gains, on chip or custom hybrid balancing network selection and a flexible signaling interface. The 29C51 is intended for use with the 2952 Integrated

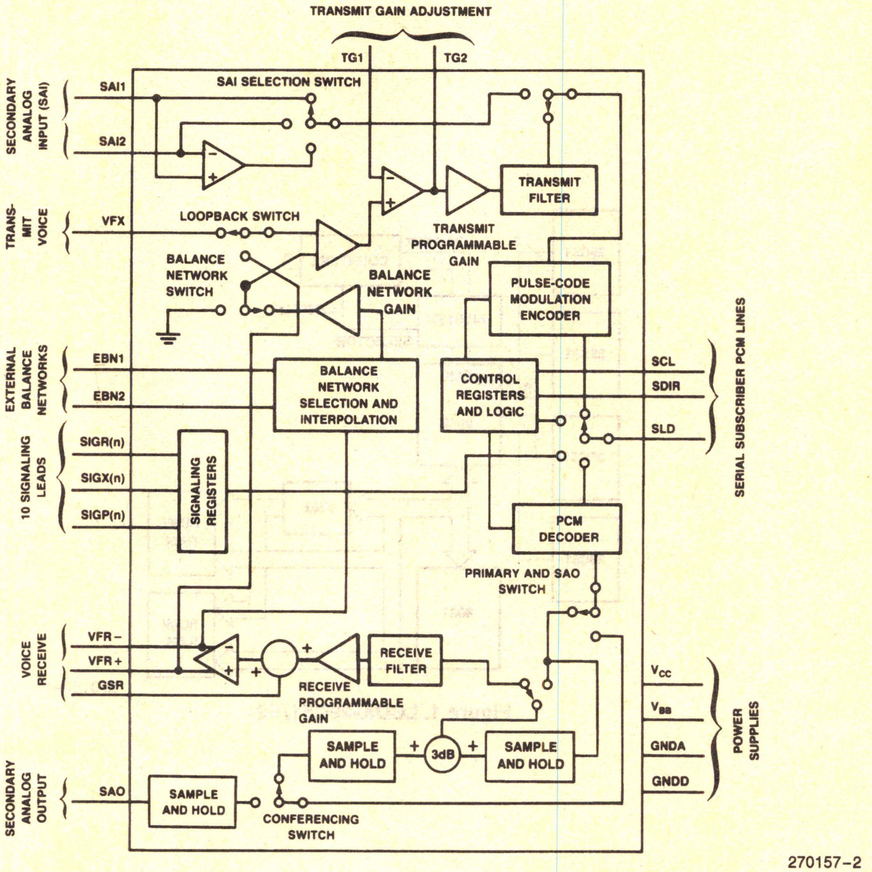


Figure 2. 29C51 Combo Block Diagram

Line Card Controller in digital switching environments. A block diagram of the 29C51 Combo is shown in Figure 2.

The 29C48 and 29C50A programmable combos allow the connection of two analog lines to one SLD line. This enables the 2952 Line Card Controller to support up to 16 subscribers on one line card.

2952 Line Card Controller

The Intel iATC 2952 Line Card Controller (LCC) is a special purpose I/O Controller optimized for use in all types of telecommunication switching systems. The 2952 replaces the traditional MSI circuits and represents the continuation of a trend to intelligent flexible line cards.

The 2952 handles the transfer of primary voice, data, feature control, and signaling information between the backplane and up to eight 29C51's; however, the LCDK-29C51/52 will only be equipped for four 29C51's per board. The 2952 and 29C51 communicate across a Subscriber Data Link (SLD) interface. The SLD is a three wire link comprising of a clock signal a serial data stream and a read/write strobe.

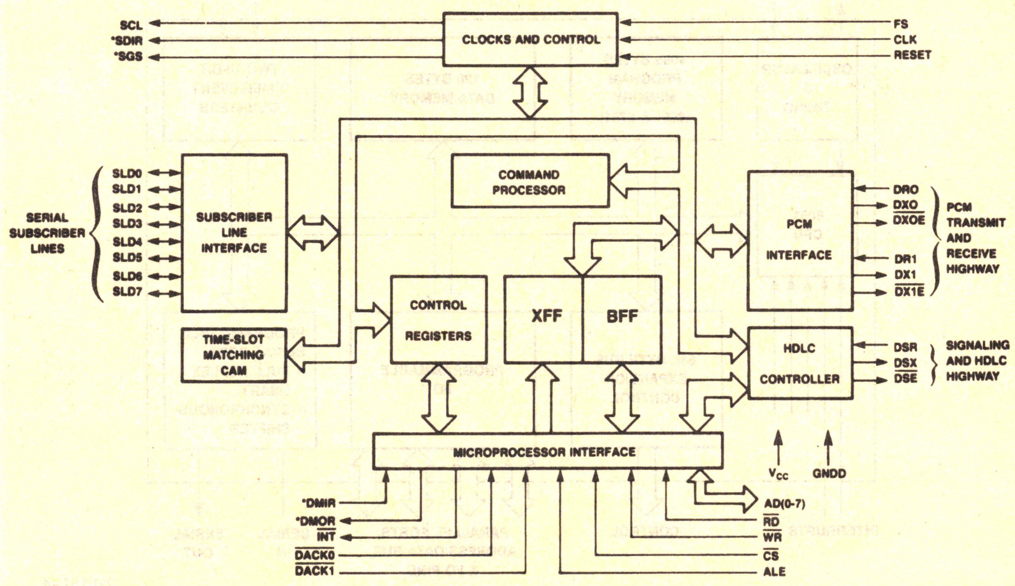
The 2952 is equipped with a standard microprocessor interface and independent transmit and receive DMA channels. As well as this, the 2952 has two standard PCM highways for connection to the backplane. Another feature of the 2952 is a fast serial interface to the central processor. This serial interface is intended for signaling and follows a subset of the HDLC protocol. Figure 3 shows a block diagram of the 2952.

System Software

A compact but powerful system monitor is contained in 8K bytes of preprogrammed ROM. The monitor includes system utilities such as command interpretation and interface controls. Table 1 summarizes the LCD-29C51/52 monitor commands.

Table 1

Command	Operation
(LCC reg)	Read or Write a 2952 register.
Half/Full	Half or Full duplex terminal mode.
Help	Displays a Help file.
Byte	Read or Write any external data memory address.
LCC	Relocation of 2952 base address.
GO	Execute user programs resident in memory.



*Dual Function Pin

Figure 3. 2952 Block Diagram

The 8031 CPU

The Intel 8031 belongs to the MCS-51 series of single chip microcomputers, and is at the heart of the LCDK, performing control and processing functions for both the primary and secondary stations. The 8031 CPU combines on a single chip: a 128 x 8 data RAM, 32 input/output lines, two 16-bit timer/event counters, a five source level nested interrupt structure, a programmable serial I/O port; and an on chip oscillator and clock circuits. An 8031 block diagram is shown in Figure 4.

For additional information on the 8031 see the 8051 User's Manual.

Memory

The memory is mapped via 4K byte pages; a maximum of 12K bytes are used allowing a user expansion of memory up to 64K bytes. The memory can be configured as 8K bytes of ROM/PROM plus 2K bytes of RAM or alternatively 12K bytes of ROM/PROM.

User Interface

Communication with the outside world is accomplished over an RS-232-C compatible link. This serial link will hook up a CRT terminal to the serial port on the 8031. Alternatively an Intellec Development System can be connected to the LCDK, this can now be used as either a dumb terminal or to transport user developed programs to the LCDK, commands for these functions are shown in Table 2. A large area of the board is laid out as general purpose wire-wrap for user custom interface.

Table 2

Command	Operation
DTERM	Places Intellec development system in dumb terminal mode.
Control-D	Transports user developed programs from Intellec development system to the LCDK.

Assembly

The LCDK-29C51/52 is supplied fully assembled and is ready to go upon power up and terminal connection. The monitor is initialized by twice typing the return key on user terminal.

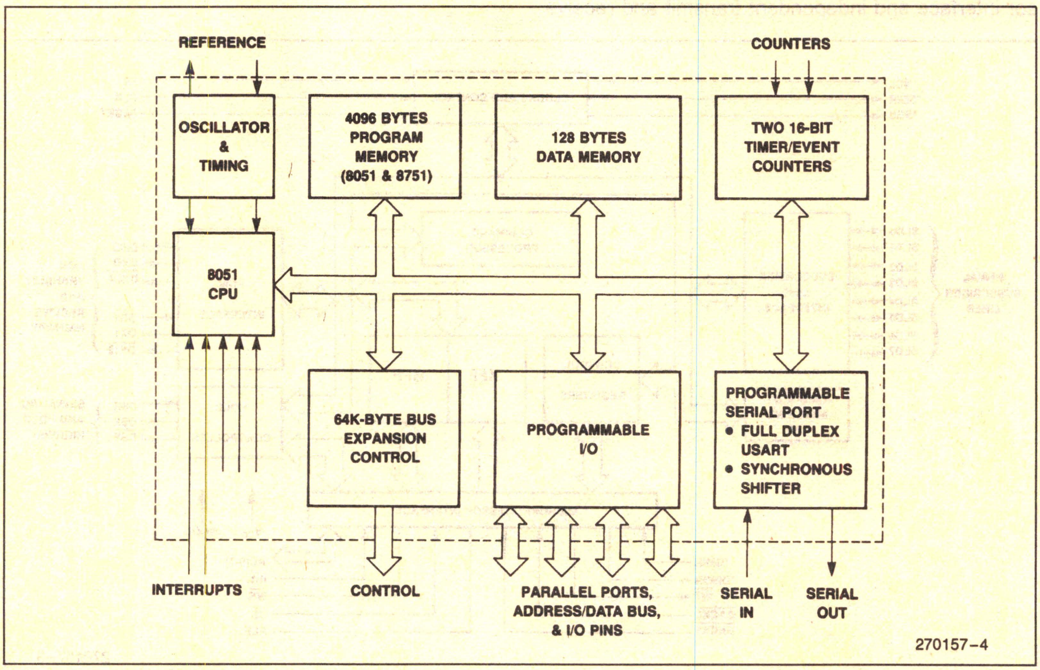


Figure 4. 8031 Block Diagram

Documentation

In addition to detailed information on using the monitor, the LCDK-29C51/52 user's manual provides circuit diagrams, a monitor listing, and a description of how the system works. The complete design library for the LCDK-29C51/52 is shown in Figure 4 and listed in the specification section under Reference Manuals.

SPECIFICATIONS

Control Processor

Intel 8031 microcomputer.
12 MHz clock rate

Memory

ROM— Socket for 256K bytes of program memory, however, a 4K or 2K byte ROM may be inserted.

RAM— Socket for 2K bytes static RAM; user configurable as program or data memory. Alternatively a 4K EPROM such as 2732A may be inserted.

Feature Control Combo

Sockets for four Intel iATC 29C51's or 29C50A. Adaptors for 29C48.

Line Card Controller

Intel iATC 2952.
User selectable 2.048 MHz or 4.096 MHz clock rate.

Interface

Ten line ribbon cable for interconnecting the primary and secondary boards, all signals are TTL compatible. Serial RS-232-C compatible interface for a terminal. Terminal baud rate can be 300, 600, 1200, 1800, 2400, 3600 or 4800.

Software

System Monitor—Preprogrammed 2732 ROM.

Monitor I/O—CRT or Intellec Development System.

Physical Characteristics

Width: 12 inches [approx. 305 mm]
Height: 1⅝ inches [approx. 41 mm]
Depth: 7⅛ inches [approx. 29 mm]
Weight: 0.864 lbs. [approx. 392 g]

Mounting

The two board may be:

— plugged into an Intel system rack
— or mounted on five horizontal legs.

Electrical Characteristics

DC Power Requirements:

Voltage	Current
±5V ±5%	2.5A
−5V ±5%	80 mA

Environmental Characteristics

Operating Temperature: 0°C–50°C

LEK29C53
29C53 LINE CARD EVALUATION KIT

- **Single Board Evaluation Kit Supporting Four 29C53 Transceivers**
- **On-Board Monitor for Access of 29C53 and 2952 from a Terminal**
- **Large User Breadboard Area**
- **Comprehensive User's Manual**
- **Compatible with LCDK29C51/52**
- **Line Interface Provided Including Pulse Transformer**

The LEK29C53 comes fully assembled, and contains the components necessary for the evaluation of the 29C53 in linecard applications. The kit allows evaluation of point-to-point as well as multipoint operation of the 29C53 when used with the 29C53 Terminal Evaluation Kit (TEK29C53). A monitor program, included in a preprogrammed ROM, allows the user to access the registers of the 29C53 transceivers, as well as those of the 2952 linecard controller, using simple English commands. Access to the monitor is via an RS232C compatible link, allowing any start-stop terminal or computer system with such a physical interface to be used with the LEK29C53. The LEK29C53 includes sockets for four 29C53 transceivers. Two are supplied with the kit. This highly flexible kit has been designed to provide a rapid introduction to the 29C53, thereby leading to shorter development time and increased productivity.

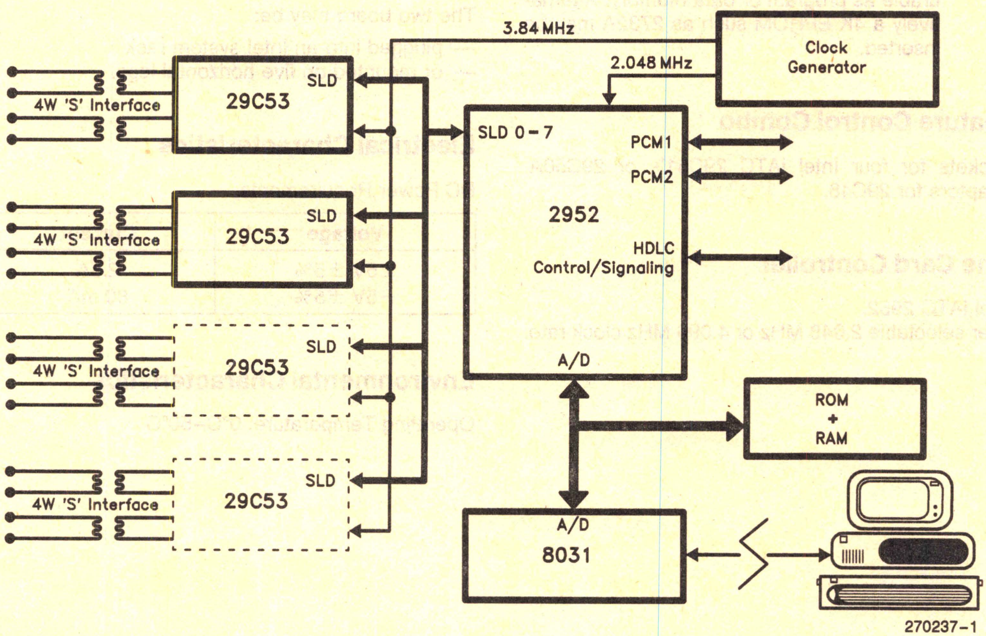


Figure 1. 29C53 Linecard Evaluation Kit

September 1986
Order Number: 270237-001

FUNCTIONAL DESCRIPTION

Overview

The LEK29C53 is a single board evaluation kit which contains the 2952 linecard controller, 8031 microcontroller, ROM including monitor, user RAM, clock generation circuitry, and sockets for four 29C53 transceivers (two devices supplied). In addition, a large breadboard area is provided for implementation of the loop interface, and any other custom circuits. The LEK29C53 can be used with the LCDK29C51/52 Primary card for full evaluation of the 2952 linecard controller, and with the TEK29C53 for full evaluation of the 29C53 in point-to-point and multipoint applications. The LEK29C53 can also be operated independently, with two of the four transceiver sockets configurable as loop slaves, allowing evaluation of S interface performance with just one board. A block diagram of the LEK29C53 is given in Figure 1.

Software

A system monitor is provided in the preprogrammed EPROM. Simple commands allow access to the registers of the 2952 and 29C53 devices. The monitor also provides for the down loading of 8031 HEX code to the onboard RAM, which can be operated as program memory for testing user code. A sample program is included in EPROM and can be run by command from the terminal interface.

User Interface

The serial port on the 8031 microcontroller is used to establish a serial communication link to a terminal for interaction with the onboard monitor program. This interface can be modified by changing jumpers to support standard DTE or DCE configurations.

Documentation

The LEK29C53 User's Manual provides a hardware description, description of monitor commands, circuit diagram, monitor listing and sample program listing. A 29C53 Reference Manual is also included with the kit.

SPECIFICATIONS

Control Processor:

Intel 8031 microcontroller.
12 MHz clock rate.

Memory

ROM - Socket accepts JEDEC pinouts to 32K X 8.

RAM - Socket will accept 8K X 8 static RAM (provided), or JEDEC pinouts to 32K X 8 for ROM.

Clock Generation:

Selectable 2.048 MHz or 4.096 MHz system clock.
Provision for external system clock.
3.84 MHz clock for 29C53.

Terminal Interface:

25 pin D type connector.
300 to 4800 Baud.

Digital Loop Controller (29C53):

Sockets for four Intel iATC 29C53 transceivers.
All four configured as loop masters, or two as loop masters, two as loop slaves.

Line Card Controller:

Intel 2952 linecard controller.

Physical and Electrical Characteristics:

Width: 12 in.(30 cm)
Height: 1 5/8 in.(4 cm)
Depth: 7 in.(18 cm)

Power requirements: 5V ± 5% @ 2.5 A
± 12V generated onboard by a DC to DC converter

Operating temperature: 10 to 40° C

ORDERING INFORMATION

29C53 Linecard Evaluation Kit, including manual — LEK29C53

Related Products:

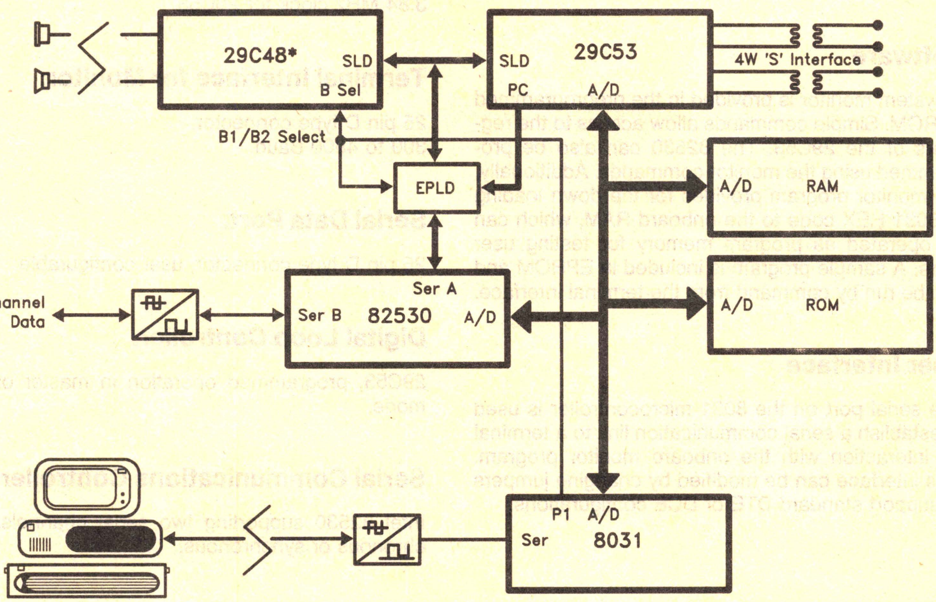
29C53 Terminal Evaluation Kit, including manual — TEK29C53

Line Card Development Kit, including manual — LCDK29C51/52

TEK29C53
29C53 TERMINAL EVALUATION KIT

- **Single Board Supports Evaluation of 29C53 Transceiver in Terminal and Terminal Adapter Applications**
- **Onboard Monitor for Access of 29C53 from a Terminal**
- **Line Interface Provided Including Pulse Transformer**
- **Includes Programmable Combo and 82530 for Voice and Data Applications**
- **Comprehensive User's Manual**
- **Large User Breadboard Area**

The TEK29C53 comes fully assembled, and contains the components necessary for the evaluation of the 29C53 in terminal and terminal adapter applications. The kit allows evaluation of point-to-point as well as multipoint operation of the 29C53 when used with the 29C53 Linecard Evaluation Kit (LEK29C53). Alternatively, the TEK29C53 can be configured to communicate directly with other TEK29C53 kits. A programmable combo is supplied with the kit for evaluation of applications supporting voice. A monitor program, included in a preprogrammed EPROM, allows the user to access the registers of the 29C53 transceiver using simple English commands. Access to the monitor is via an RS232C compatible link, allowing any start-stop terminal or computer system with such a physical interface to be used with the TEK29C53. This highly flexible ISDN terminal evaluation kit has been designed to provide a rapid introduction to the 29C53, thereby leading to shorter development time and increased productivity.



* Sockets are available for both the 29C48 and the 29C50A

Figure 1. 29C53 Terminal Evaluation Kit

September 1986
Order Number: 270236-001

FUNCTIONAL DESCRIPTION

Overview

The TEK29C53 is a single board evaluation kit which contains the 29C53 transceiver, 8031 microcontroller, EPROM including monitor, user RAM, clock generation circuitry, and 82530 serial communications controller. In addition, a large breadboard area is provided for implementation of the loop interface, phone interface, or any other customer circuits. A programmable SLD combo, interfaced through the SLD port of the 29C53, provides access to one of the B channels of the S interface for applications supporting voice. The 82530 communications controller provides access to the second B channel, also via the SLD port of the 29C53, for packet data transfers. The second channel of the 82530 can be used as an interface to synchronous or asynchronous terminals. For full evaluation of the 29C53 in point-to-point and multipoint applications, the LEK29C53 Linecard Evaluation Kit can be used with the TEK29C53, or the TEK29C53 can itself be configured as a loop master for communication with one or more other TEK29C53 boards. A block diagram of the TEK29C53 is given in Figure 1.

Software

A system monitor is provided in the preprogrammed EPROM. Simple commands allow access to the registers of the 29C53. The 82530 can also be programmed using the monitor commands. Additionally, the monitor program provides for the down loading of 8031 HEX code to the onboard RAM, which can be operated as program memory for testing user code. A sample program is included in EPROM and can be run by command from the terminal interface.

User Interface

The serial port on the 8031 microcontroller is used to establish a serial communication link to a terminal for interaction with the onboard monitor program. This interface can be modified by changing jumpers to support standard DTE or DCE configurations.

Documentation

The TEK29C53 User's Manual provides a hardware description, description of monitor commands, circuit diagram, monitor listing and sample program listing. A 29C53 Reference Manual is also included with the kit.

SPECIFICATIONS

Control Processor:

Intel 8031 microcontroller.
12 MHz clock rate.

Memory:

ROM - Socket accepts JEDEC pinouts to 32K X 8.

RAM - Socket will accept 8K X 8 static RAM (provided), or JEDEC pinouts to 32K X 8 for ROM.

Clock Generation:

3.84 MHz clock for 29C53.

Terminal Interface for Monitor:

25 pin D type connector.
300 to 4800 Baud.

Serial Data Port:

25 pin D type connector, user configurable.

Digital Loop Controller:

29C53, programmed operation in master or slave mode.

Serial Communications Controller:

Intel 82530 supporting two serial channels, asynchronous or synchronous.

Feature Control Combo

Intel 29C48 with additional socket available for the Intel 29C50A.

Physical and Electrical Characteristics:

Width: 12 in.(30 cm)
Height: 2 in.(5 cm)
Depth: 7 in.(18 cm)

Power requirements: 5V ± 5% @ 2.5 A
−5V, ± 12V generated on-board by a DC to DC converter

Operating temperature: 10 to 40° C

ORDERING INFORMATION

29C53 Terminal Evaluation Kit, including manual — TEK29C53

Related Products:

29C53 Linecard Evaluation Kit, including manual — LEK29C53

Line Card Development Kit, including manual — LCDK29C51/52

intel®

APPLICATION NOTE

AP-255

October 1986

Military CHMOS: Designing With the M80C51BH

TOM WILLIAMSON
MCO APPLICATIONS ENGINEER

Order Number: 270081-001

CMOS EVOLVES

The original CMOS logic families were the 4000-series and the 74C-series circuits. The 74C-series circuits are functional equivalents to the correspondingly numbered 74-series TTL circuits, but have CMOS logic levels and retain the other well known characteristics of CMOS logic.

These characteristics are: low power consumption, high noise immunity, and slow speed. The low power consumption is inherent to the nature of the CMOS circuit. The noise immunity is due partly to the CMOS logic levels, and partly to the slowness of the circuits. The slow speed is due to the technology used to construct the transistors in the circuit.

The technology used is called metal-gate CMOS, because the transistor gates are formed by metal deposition. More importantly, the gates are formed after the drain and source regions have been defined, and must overlap the source and drain somewhat to allow for alignment tolerances. This overlap plus the relatively large size of the transistors themselves result in high electrode capacitance, and that is what limits the speed of the circuit.

High speed CMOS became feasible with the development of the self-aligning silicon gate technology. In this process polysilicon gates are deposited **before** the source and drain regions are defined. Then the source and drain regions are formed by ion implantation using the gate itself as a mask for the implantation. This eliminates most of the overlap capacitance. In addition, the process allows smaller transistors. The result is a significant increase in circuit speed. The 74HC-series of CMOS logic circuits is based on this technology, and has speeds comparable to LS TTL, which is to say about 10 times faster than the 74C-series circuits.

The size reduction that contributes to the higher speed also demands an accompanying reduction in the maximum supply voltage. High-speed CMOS is generally limited to 6V.

WHAT IS CHMOS?

CHMOS is the name given to Intel's high-speed CMOS processes. There are two CHMOS processes, one based on an n-well structure and one based on a p-well structure. In the n-well structure, n-type wells are diffused into a p-type substrate. Then the n-channel transistors (nFETs) are built into the substrate and pFETs are built into the n-wells. In the p-well structure, p-type wells are diffused into an n-type substrate. Then the nFETs are built into the wells and pFETs, into the substrate. Both processes have their advantages and disadvantages, which are largely transparent to the user.

Lower operating voltages are easier to obtain with the p-well structure than with the n-well structure. But the p-well structure does not easily adapt to an EPROM which would be pin-for-pin compatible with HMOS EPROMs. On the other hand the n-well structure can be based on the solidly founded HMOS process, in which nFETs are built into a p-type substrate. This allows somewhat more than half of the transistors in a CHMOS chip to be constructed by processes that are already well characterized.

Currently Intel's CHMOS microcontrollers and memory products are n-well devices, whereas CHMOS microprocessors are p-well devices.

Another benefit of CHMOS III, that is of particular interest to military designers, is its radiation tolerance. Intel's CHMOS III process is based on the proven radiation tolerance of HMOS III. Preliminary data shows total dose tolerances in excess of 100 KRads (si). For information on radiation tolerance, see reference #8.

Further discussion of the CHMOS technology is provided in references 1 and 2 (which are reprinted in the Microcontroller Handbook).

THE MCS®-51 FAMILY IN CHMOS

The M80C51BH is the CHMOS version of Intel's original M8051. The M80C31BH is the ROMless M80C51BH, equivalent to the M8031. These CHMOS devices are architecturally identical with their HMOS counterparts, except that they have two added features for reduced power. These are the Idle and Power Down modes of operation.

In most cases, an M80C51BH can directly replace the M8051 in existing applications. It can execute the same code at the same speed, accept signals from the same sources, and drive the same loads. However, the M80C51BH covers a wider range of speeds, will emit CMOS logic levels to CMOS loads, and will draw about 1/10 the current of an M8051 (and less yet in the reduced power modes). Interchangeability between the HMOS and CHMOS devices is discussed in more detail in the final section of this Application Note.

It should be noted that the M80C51BH CPU is not static. That means if the clock frequency is too low, the CPU might forget what it was doing. This is because the circuitry uses a number of dynamic nodes. A dynamic node is one that uses the node-to-ground capacitance to form a temporary storage cell. Dynamic nodes are used to reduce the transistor count, and hence the chip area, thus to produce a more economical device.

This is not to say that the on-chip RAM in CHMOS microcontrollers is dynamic. It's not. It's the CPU that is dynamic, and that is what imposes the minimum clock frequency specification.

LATCHUP

Latchup is an SCR-type turn-on phenomenon that is the traditional nemesis of CMOS systems. The substrate, the wells, and the transistors from parasitic pnpn structures within the device. These parasitic structures turn on like an SCR if a sufficient amount of forward current is driven through one of the junctions. From the circuit designer's point of view it can happen whenever an input or output pin is externally driven a diode drop above V_{CC} or below V_{SS}, by a source that is capable of supplying the required trigger current.

However much of a problem latchup has been in the past, it is good to know that in most recently developed CMOS devices, and specifically in CHMOS devices, the current required to trigger latchup is typically well over 100 mA. The M80C51BH is virtually immune to latchup. (References 1 and 2 present a discussion of the latchup mechanisms and the steps that are taken on the chip to guard against it.) Modern CMOS is not absolutely immune to latchup, but with trigger currents in the hundreds of mA, latchup is certainly a lot easier to avoid than it once was.

A careless power-up sequence might trigger a latchup in the older CMOS families, but it's unlikely to be a major problem in high-speed CMOS or in CHMOS. There is still some risk incurred in inserting or removing chips or boards in a CMOS system while the power is on. Also, severe transients, such as inductive kicks or momentary short-circuits, can exceed the trigger current for latchup.

For applications in which some latchup risk seems unavoidable, you can put a small resistor (100Ω or so) in series with signal lines to ensure that the trigger current will never be reached. This also helps to control overshoot and RFI.

LOGIC LEVELS AND INTERFACING PROBLEMS

CMOS logic levels differ from TTL levels in two ways. First, for equal supply voltages, CMOS gives (and requires) a higher "logic 1" level than TTL. Secondly, CMOS logic levels are V_{CC} (or VDD) dependent, whereas guaranteed TTL logic levels are fixed when V_{CC} is within TTL specs.

Standard 74HC logic levels are as follows:

$$V_{IH}MIN = 70\% \text{ of } V_{CC}$$
$$V_{IL}MAX = 20\% \text{ of } V_{CC}$$
$$V_{OH}MIN = V_{CC} - 0.1V, |I_{OH}| \leq 20\ \mu A$$
$$V_{OL}MAX = 0.1V, |I_{OL}| \leq 20\ \mu A$$

Figure 1 compares 74HC, LS TTL, and 74HCT logic levels with those of the HMOS M8051 and the CHMOS M80C51BH for V_{CC} = 5V.

Output logic levels depend of course on load current, and are normally specified at several load currents. When CMOS and TTL are powered by the same V_{CC}, the logic levels guaranteed on the data sheets indicate that CMOS can drive TTL, but TTL can't drive CMOS. The incompatibility is that the TTL circuit's V_{OH} level is too low to reliably be recognized by the CMOS circuit as a valid V_{IH}.

Since HMOS circuits were designed to be TTL-compatible, they have the same incompatibility.

Fortunately, 74HCT-series circuits are available to ease these interfacing problems. They have TTL-compatible logic levels at the inputs and standard CMOS levels at the outputs.

The M80C51BH is designed to work with either TTL or CMOS. Therefore its logic levels are specified very much like 74HCT circuits. That is, its input logic levels are TTL-compatible, and its output characteristics are like standard high-speed CMOS.

NOISE CONSIDERATIONS

One of the major reasons for going to CMOS has traditionally been that CMOS is less susceptible to noise. As previously noted, its low susceptibility to noise

Logic State	V_{CC} = 5V				
	74HC	74HCT	LS TTL	M8051	M80C51BH
V_{IH}	3.5V	2.0V	2.0V	2.0V	1.9V
V_{IL}	1.0V	0.8V	0.8V	0.8V	0.9V
V_{OH}	4.9V	4.9V	2.7V	2.4V	4.5V
V_{OL}	0.1V	0.1V	0.5V	0.45V	0.45V

Figure 1. Logic Level Comparison. (Output voltage levels depend on load current. Data sheets list guaranteed output levels for several load currents. The output levels listed here are for minimum loading.)

is partly due to superior noise margins, and partly due to its slow speed.

Noise margin is the difference between V_{OL} and V_{IL}, or between V_{OH} and V_{IH}. If V_{OH} from a driving circuit is 2.7V and V_{IH} to the driven circuit is 2.0V, then the driven circuit has 0.7V of noise margin at the logic high level. These kinds of comparisons show that an all-CMOS system has wider noise margins than an all-TTL system.

Figure 2 shows noise margins in CMOS and LS TTL systems when both have $V_{CC} = 5V$. It can be seen that CMOS/CMOS and CMOS/CHMOS systems have an edge over LS TTL in this respect.

Noise margins can be misleading, however, because they don't say how much noise energy it takes to induce in the circuit a noise voltage of sufficient amplitude to cause a logic error. This would involve consideration of the width of the noise pulse as compared with the circuit's response speed, and the impedance to ground from the point of noise introduction in the circuit.

When these considerations are included, it is seen that using the slower 74C- and 4000-series circuits with a 12 or 15 volt supply voltage does offer a truly improved level of noise immunity, but that high-speed CMOS at 5V is not significantly better than TTL.

One should not mistake the wider supply voltage tolerance of high-speed CMOS for V_{CC} glitch immunity. Supply voltage tolerance is a DC rating, not a glitch rating.

For any clocked CMOS, and most especially for VLSI CMOS, V_{CC} decoupling is critical. CHMOS draws current in extremely sharp spikes at the clock edges. The VHF and UHF components of these spikes are not

Interface	Noise Margin for $V_{CC} = 5V$	
	Logic Low $V_{IL}-V_{OL}$	Logic High $V_{OH}-V_{IH}$
74HC to 74HC	0.9V	1.4V
LSTTL to LSTTL	0.3V	0.7V
LSTTL to 74HCT	0.3V	0.7V
LSTTL to M80C51BH	0.3V	0.7V
74HC to M80C51BH	0.8V	3.0V
M80C51BH to 74HC	0.8V	1.0V

Figure 2. Noise Margins for CMOS and LS TTL Circuits

drawn from the power supply, but from the decoupling capacitor. If the decoupling circuit is not sufficiently low in inductance, V_{CC} will glitch at each clock edge. We suggest that a 0.1 μF decoupler cap be used in a minimum-inductance configuration with the microcontroller. A minimum-inductance configuration is one that minimizes the area of the loop formed by the chip (V_{CC} and V_{SS}), the traces to the decoupler cap, and the decoupler cap. PCB designers too often fail to understand that if the traces that connect the decoupler cap to the V_{CC} and V_{SS} pins aren't short and direct, the decoupler loses much of its effectiveness.

Overshoot and ringing in signal lines are potential sources of logic upsets. These can largely be controlled by circuit layout. Inserting small resistors (about 100Ω) in series with signal lines that seem to need them will also help.

The sharp edges produced by high-speed CMOS can cause RFI problems. The severity of these problems is largely a function of the PCB layout. We don't mean to imply that all RFI problems can be solved by a better PCB layout. It may well be, for example, that in some RFI-sensitive designs high-speed CMOS is simply not the answer. But circuit layout is a critical factor in the noise performance of any electronic system, and more so in high-speed CMOS systems than others.

Circuit layout techniques for minimizing noise susceptibility and generation are discussed in references 3 through 6.

UNUSED PINS

CMOS input pins should not be left to float, but should always be pulled to one logic level or the other. If they float, they tend to float into the transition region between 0 and 1, where the pullup and pulldown devices in the input buffer are both conductive. This causes a significant increase in I_{CC}. A similar effect exists in HMOS circuits, but with less noticeable results.

In M80C51BH and M80C31BH designs, unused pins of Ports 1, 2, and 3 can be ignored, because they have internal pullups that will hold them at a valid Logic 1 level. Port 0 pins are different, however, in not having internal pullups (except during bus operations).

When M80C51BH is in reset, the Port 0 pins are in a float state unless they are externally pulled up or down. If it's going to be held in reset for just a short time, the transient float state can probably be ignored. When it comes out of reset, the pins stay afloat unless they are externally pulled either up or down. Alternatively, the software can internally write 0s to whatever Port 0 pins may be unused.

The same considerations are applicable to the M80C31BH with regards to reset. But when the M80C31BH comes out of reset, it commences bus operations, during which the logic levels at the pins are always well defined as high or low.

Consider the M80C31BH in the Power Down or Idle modes, however. In those modes it is not fetching instructions, and the Port 0 pins will float if not externally pulled high or low. The choice of whether to pull them high or low is the designer's. Normally it is sufficient to pull them up to V_{CC} with 10K resistors. But if power is going to be removed from circuits that are connected to the bus, it will be advisable to pull the bus pins down (normally with 10K resistors). Considerations involved in selecting pullup and pulldown resistor values are as follows.

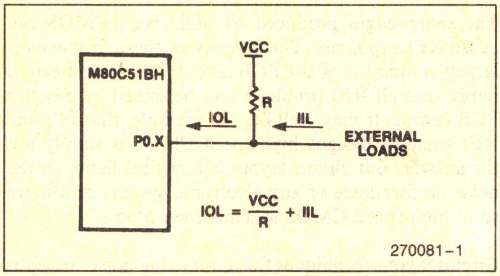


Figure 3a. Conditions defining the minimum value for R. P0.X is emitting a logic low. R must be large enough to not cause I_{OL} to exceed data sheet specifications.

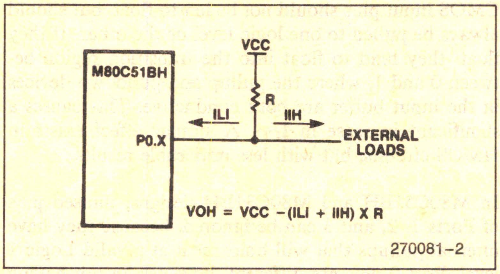


Figure 3b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep V_{OH} acceptably high.

PULLUP RESISTORS

If a pullup resistor is to be used on a Port 0 pin, its minimum value is determined by I_{OL} requirements. If the pin is trying to emit a 0, then it will have to sink the current from the pullup resistor plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 3a, while maintaining a valid output low (V_{OL}). To guarantee that the pin voltage will not exceed 0.45V, the resistor should be selected so that I_{OL} doesn't exceed the value specified on the data sheet. In most CMOS applications, the minimum value would be about 2 KΩ.

The maximum value you could use depends on how fast you want the pin to pull up after bus operations have ceased, and how high you want the V_{OH} level to be. The smaller the resistor the faster it pulls up. Its effect on the V_{OH} level is that $V_{OH} = V_{CC} - (I_{LI} + I_{IH}) \times R$. I_{LI} is the input leakage current to the Port 0 pin, and I_{IH} is the input high current to the external loads, as shown in Figure 3b. Normally V_{OH} can be expected to reach 0.9 V_{CC} if the pullup resistance does not exceed about 50 KΩ.

PULLDOWN RESISTORS

If a pulldown resistor is to be used on a Port 0 pin, its minimum value is determined by V_{OH} requirements during bus operations, and its maximum value is in most cases determined by leakage current.

During bus operations the port uses internal pullups to emit 1s. The D.C. Characteristics in the data sheet list guaranteed V_{OH} levels for given I_{OH} currents. (The "-" sign in the I_{OH} value means the pin is sourcing that current to the external load, as shown in Figure 4.) To ensure the V_{OH} level listed in the data sheet, the re-

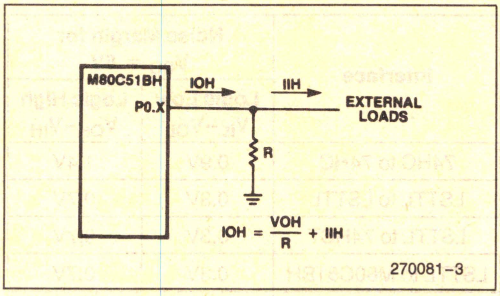


Figure 4a. Conditions defining the minimum value for R. P0.X is emitting a 1 in a bus operation. R must be large enough to not cause I_{OH} to exceed data sheet specifications.

sistor has to satisfy

$$\frac{V_{OH}}{R} + I_{IH} \leq |I_{OH}|$$

where I_{IH} is the input high current to the external loads.

When the pin goes into a high impedance state, the pulldown resistor will have to sink leakage current from the pin, plus whatever other current may be sourced by other loads connected to the pin, as shown in Figure 4b. The Port 0 leakage current is I_{LI} on the data sheet. The resistor should be selected so that the voltage developed across it by these currents will be seen as a logic low by whatever circuits are connected to it (including the M80C51BH). In CMOS/CHMOS applications, 50 KΩ is normally a reasonable maximum value.

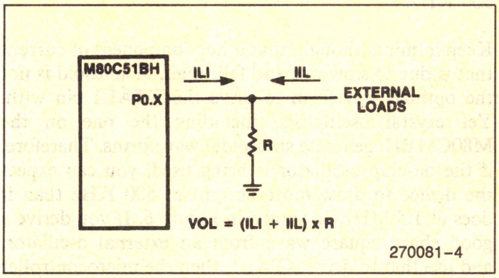


Figure 4b. Conditions defining the maximum value for R. P0.X is in a high impedance state. R must be small enough to keep V_{OL} acceptably low.

DRIVE CAPABILITY OF THE INTERNAL PULLUPS

There's an important difference between HMOS and CHMOS port drivers. The pins of Ports 1, 2, and 3 of the CHMOS parts each have three pullups: strong, normal, and weak, as shown in Figure 5. The strong pullup (p1) is only used during 0-to-1 transitions, to hasten the transition. The weak pullup (p2) is on whenever the bit latch contains a 1. The "normal" pullup (p3) is controlled by the pin voltage itself.

The reason that p3 is controlled by the pin voltage is that if the pin is being used as an input, and the external source pulls it to a low, then turning off p3 makes for a lower I_{IL}. The data sheet shows an "I_{TL}" specification. This is the current that p3 will source during the time the pin voltage is making its 1-to-0 transition. This is what I_{IL} would be if an input low at the pin didn't turn p3 off.

Note, however, that this p3 turn-off mechanism puts a restriction on the drive capacity of the pin if it's being used as an output. If you're trying to output a logic high, and the external load pulls the pin voltage below the pin's V_{IH}MIN spec, p3 might turn off, leaving only the weak p2 to provide drive to the load. To prevent this happening, you need to ensure that the load doesn't draw more than the I_{OH} spec for a valid V_{OH}. The idea is to make sure the pin voltage never falls below its own V_{IH}MIN specification.

POWER CONSUMPTION

The main reason for going to CMOS, of course, is to conserve power. (There are other reasons, but this

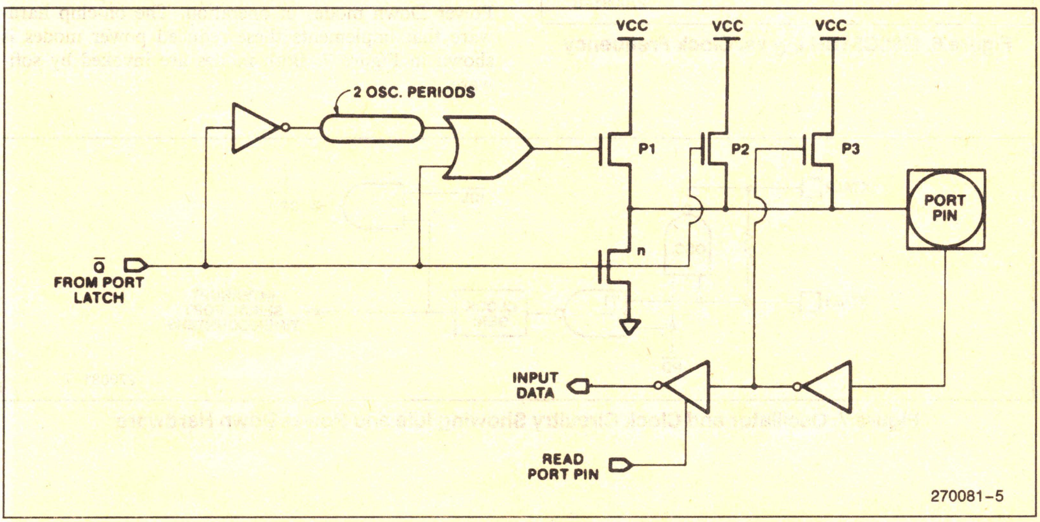


Figure 5. M80C51BH Output Drivers for Ports 1, 2 and 3

is the main one.) Conserving power doesn't mean just reducing your electric bill. Nor does it necessarily relate to battery operation, although battery operation without CMOS is pretty unhandy. The main reason for conserving power is to be able to put more functionality into a smaller space. The reduced power consumption allows the use of smaller and lighter power supplies, and less heat being generated allows denser packaging of circuit components. Expensive fans and blowers can usually be eliminated.

A cooler running chip is also more reliable, since most random and wearout failures relate to die temperature. And finally, the lower power dissipation will allow more functions to be integrated onto the chip.

The reason CMOS consumes less power than NMOS is that when it's in a stable state there is no path of conduction from V_{CC} to V_{SS} except through various leakage paths. CMOS does draw current when it's changing states. How much current it draws depends on how often and how quickly it changes states.

CMOS circuits draw current in sharp spikes during logical transitions. These current spikes are made up of two components. One is the current that flows during the transition time when pullup and pulldown FETs are both active. The average (DC) value of this component is larger when the transition times of the input signals are longer. For this reason, if the current draw is a critical factor in the design, slow rise and fall times should be avoided, even when the system speed doesn't seem to justify a need for nanosecond switching speeds.

The other component is the current that charges stray and load capacitance at the nodes of a CMOS logic gate. The average value of this current spike is its area (integral over time) multiplied by its rep rate. Its area is the amount of charge it takes to raise the node capacitance, C, to V_{CC}. That amount of charge is just $C \times V_{CC}$. So the average value of the current spike is $C \times V_{CC} \times f$, where f is the clock frequency.

This component of current increases linearly with clock frequency. For minimal current draw, the M80C51BH-2 is spec'd to run at frequencies as low as 500 KHz.

Keep in mind, though, that other component of current that is due to slow rise and fall times. A sinusoid is not the optimal waveform to drive the XTAL1 pin with. Yet crystal oscillators, including the one on the M80C51BH, generate sinusoidal waveforms. Therefore, if the on-chip oscillator is being used, you can expect the device to draw more current at 500 KHz than it does at 1.5 MHz, as shown in Figure 6. If you derive a good sharp square wave from an external oscillator, and use that to drive XTAL1, then the microcontroller will draw less current. But the external oscillator will probably make up the difference.

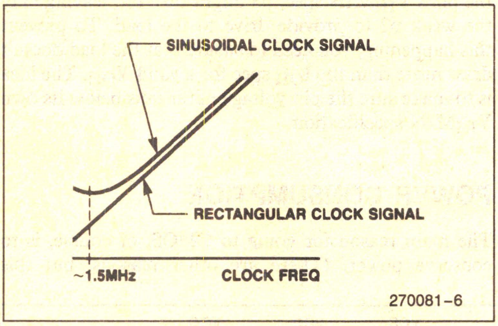


Figure 6. M80C51BH I_{CC} vs. Clock Frequency

The M80C51BH has two power-saving features not available in the HMOS devices. These are the Idle and Power Down modes of operation. The on-chip hardware that implements these reduced power modes is shown in Figure 7. Both modes are invoked by software.

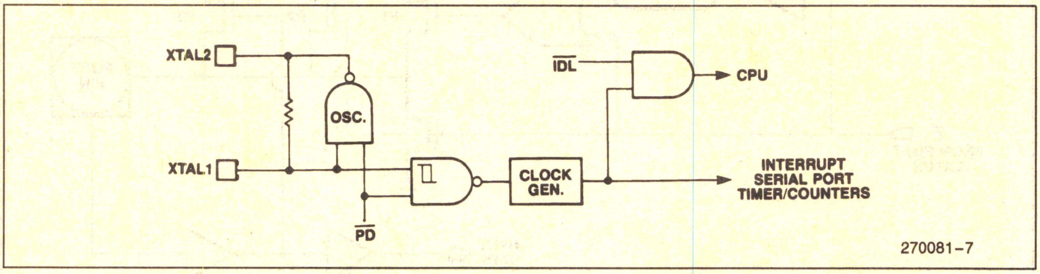


Figure 7. Oscillator and Clock Circuitry Showing Idle and Power Down Hardware

Idle

In the Idle Mode ($\overline{IDL}$ = 0 in Figure 7), the CPU puts itself to sleep by gating off its own clock. It doesn't stop the oscillator. It just stops the internal clock signal from getting to the CPU. Since the CPU draws 80 to 90 percent of the chip's power, shutting it off represents a fairly significant power savings. The on-chip peripherals (timers, serial port, interrupts, etc.) and RAM continue to function as normal. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

The Idle Mode is invoked by setting bit 0 (IDL) of the PCON register. PCON is not bit-addressable, so the bit has to be set by a byte operation, such as

```
ORL     PCON,#1
```

The PCON register also contains flag bits GF0 and GF1, which can be used for any general purposes, or to give an indication if an interrupt occurred during normal operation or during Idle. In this application, the instruction that invokes Idle also sets one or both of the flag bits. Their status can then be checked in the interrupt routines.

While the device is in the Idle Mode, ALE and $\overline{PSEN}$ emit logic high (V_{OH}), as shown in Figure 8. This is so external EPROM can be deselected and have its output disabled.

The port pins hold the logical states they had at the time the Idle was activated. If the device was executing out of external program memory, Port 0 is left in a high impedance state and Port 2 continues to emit the high byte of the program counter (using the strong pullups to emit 1s). If the device was executing out of internal program memory, Ports 0 and 2 continue to emit whatever is in the P0 and P2 registers.

There are two ways to terminate Idle. Activation of any enabled interrupt will cause the hardware to clear bit 0 of the PCON register, terminating the Idle Mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that invoked Idle.

The other way is with a hardware reset. Since the clock oscillator is still running, RST only needs to be held active for two machine cycles (24 oscillator periods) to complete the reset. Note that this exit from Idle writes 1s to all the ports, initializes all SFRs to their reset values, and restarts program execution from location 0.

Power Down

In the Power Down Mode ($\overline{PD}$ = 0 in Figure 7), the CPU puts the whole chip to sleep by turning off the oscillator. In case it was running from an external oscillator, it also gates off the path to the internal phase generators, so no internal clock is generated even if the external oscillator is still running. The on-chip RAM, however, saves its data, as long as V_{CC} is maintained. In this mode the only I_{CC} that flows is leakage, which is normally in the micro-amp range.

The Power Down Mode is invoked by setting bit 1 in the PCON register, using a byte instruction such as

```
ORL       PCON,#2
```

While the device is in Power Down, ALE and $\overline{PSEN}$ emit lows (V_{OL}), as shown in Figure 8. The reason they are designed to emit lows is so that power can be removed from the rest of the circuit, if desired, while the M80C51BH is in its Power Down Mode.

The port pins continue to emit whatever data was written to them. Note that Port 2 emits its P2 register data even if execution was from external program

Pin	Internal Execution		External Execution	
	Idle	Power Down	Idle	Power Down
ALE	1	0	1	0
$\overline{PSEN}$	1	0	1	0
P0	SFR Data	SFR Data	High-Z	High-Z
P1	SFR Data	SFR Data	SFR Data	SFR Data
P2	SFR Data	SFR Data	PCH	SFR Data
P3	SFR Data	SFR Data	SFR Data	SFR Data

Figure 8. Status of Pins in Idle and Power Down Modes. "SFR Data" means the port pins emit their internal register data. "PCH" is the high byte of the Program Counter.

memory. Port 0 also emits its P0 register data, but if execution was from external program memory, the P0 register data is FF. The oscillator is stopped, and the part remains in this state as long as V_{CC} is held, and until it receives an external reset signal.

The only exit from Power Down is a hardware reset. Since the oscillator was stopped, RST must be held active long enough for the oscillator to re-start and stabilize. Then the reset function initializes all the Special Function Registers (ports, timers, etc.) to their reset values, and re-starts the program from location 0. Therefore, timer reloads, interrupt enables, baud rates, port status, etc. need to be re-established. Reset does not affect the content of the on-chip data RAM. If V_{CC} was held during Power Down, the RAM data is still good.

USING THE POWER DOWN MODE

The software-invoked Power Down feature offers a means of reducing the power consumption to a mere trickle in systems which are to remain dormant for some period of time, while retaining important data.

The user should give some thought to what state the port pins should be left in during the time the clock is stopped, and write those values to the port latches before invoking Power Down.

If V_{CC} is going to be held to the entire circuit, one would want to write values to the port latches that would deselect peripherals before invoking Power Down. For example, if external memory is being used, the P2 SFR should be loaded with a value which will not generate an active chip select to any memory device.

In some applications, V_{CC} to part of the system may be shut off during Power Down, so that even quiescent and standby currents are eliminated. Signal lines that connect to those chips must be brought to a logic low, whether the chip in question is CMOS, NMOS, or TTL, before V_{CC} is shut off to them. CMOS pins have parasitic pin junctions to V_{CC}, which will be forward biased if V_{CC} is reduced to zero while the pin is held at a logic high. NMOS pins often have FETs that look like diodes to V_{CC}. TTL circuits may actually be damaged by an input high if $V_{CC} = 0$. That's why the M80C51BH outputs lows at ALE and $\overline{\text{PSEN}}$ during Power Down.

Figure 9 shows a circuit that can be used to turn V_{CC} off to part of the system during Power Down. The circuit will ensure that the secondary circuit is not de-energized until after the M80C31BH is in Power Down, and that the M80C31BH does not receive a reset (terminating the Power Down Mode) before the secondary circuit is re-energized. Therefore, the program memory itself can be part of the secondary circuit.

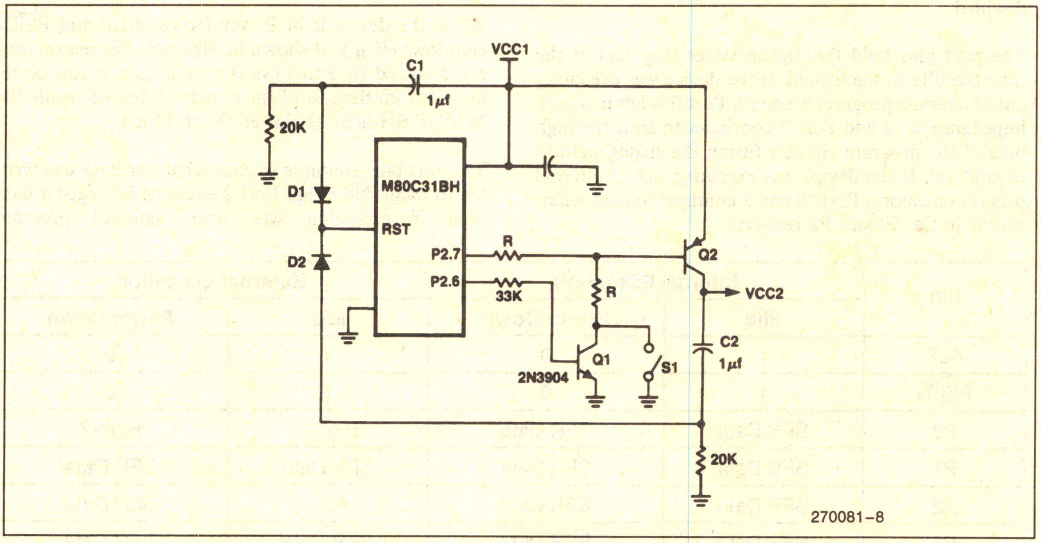
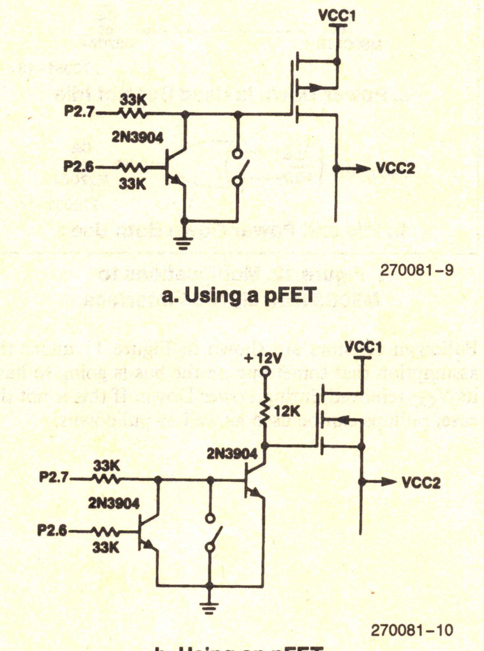


Figure 9. The M80C31BH de-energizes part of the circuit (V_{CC2}) when it goes into Power Down. Selections of R and Q2 depend on V_{CC2} current draw.

In Figure 9, when V_{CC} is switched on to the M80C31BH, capacitor C1 provides a power-on reset. The reset function writes 1s to all the port pins. The 1 at P2.6 turns Q1 on, enabling V_{CC} to the secondary circuit through transistor Q2. As the M80C31BH comes out of reset, Port 2 commences emitting the high byte of the Program Counter, which results in the P2.7 and P2.6 pins outputting 0s. The 0 at P2.7 ensures continuation of V_{CC} to the secondary circuit.

The system software must now write a 1 to P2.7 and a 0 to P2.6 in the Port 2 SFR, P2. These values will not appear at the Port 2 pins as long as the device is fetching instructions from external program memory. However, whenever the M80C31BH goes into Power Down, these values will appear at the port pins, and will shut off both transistors, disabling V_{CC} to the secondary circuit.

Closing the switch S1 re-energizes the secondary circuit, and at the same time sends a reset through C2 to the M80C31BH to wake it up. The diode D1 is to prevent C1 from hogging current from C2 during this secondary reset. D2 prevents C2 from discharging through the RST pin when V_{CC} to the secondary circuit goes to zero.



a. Using a pFET

b. Using an nFET

Figure 10. Using power MOSFETs to Control V_{CC2}

Using Power MOSFETS to Control V_{CC}

Power MOSFETs are gaining in popularity (and availability). The easiest way to control V_{CC} is with a Logic Level pFET, as shown in Figure 10a. This circuit allows the full V_{CC} to be used to turn the device on. Unfortunately, power pFETs are not economically competitive with bipolar transistors of comparable ratings.

Power nFETs are both economical and available, and can be used in this application if a DC supply of higher voltage is available to drive the gate. Figure 10b shows how to implement a V_{CC} switch using a power nFET and a (nominally) +12V supply. The problem here is that if the device is on, its source voltage is +5V. To maintain the on state, the gate has to be another 5 or 10V above that. The "12V" supply is not particuarly critical. A minimally filtered, unregulated rectifier will suffice.

M80C31BH + CHMOS EPROM

The M27C64 is Intel's 8K byte CHMOS EPROM. The M27C64 requires an external address latch, and can be used with the M80C31BH as shown in Figure 11a. In most M8031 + M2764 (HMOS) applications, the M2764's Chip Enable ($\overline{CE}$) pin is hard-wired to ground (since it's normally the only program memory on the bus). This can be done with the CHMOS versions as well, but there is some advantage in connecting $\overline{CE}$ to ALE, as shown in Figure 11a. The advantage is that if the M80C31BH is put into Idle mode, since ALE goes to a 1 in that mode, the M27C64 will be deselected and go into a low current standby mode.

The timing waveforms for this configuration are shown in Figure 11b. In Figure 11b the signals and timing parameters in parentheses are those of the M27C64, and the others are of the M80C31BH, except Tprop is a parameter of the address latch. The requirements for timing compatibility are

TAVIV - Tprop > tACC
TLLIV > tCE
TPLIV > tOE
TPXIZ > tDF

If the application is going to use the Power Down mode then we have another consideration: In Idle, ALE = $\overline{PSEN}$ = 1, and in Power Down, ALE = $\overline{PSEN}$ = 0. In a realistic application there are likely to be more chips in the circuit than are shown in Figure 11, and it is likely that the nonessential ones will have their V_{CC} removed while the CPU is in Power Down. In that case the EPROM and the address latch should be among the chips that have V_{CC} removed, and logic lows are exactly what are required at ALE and $\overline{PSEN}$.

But if V_{CC} is going to be maintained to the EPROM during Power Down, then it will be necessary to de-

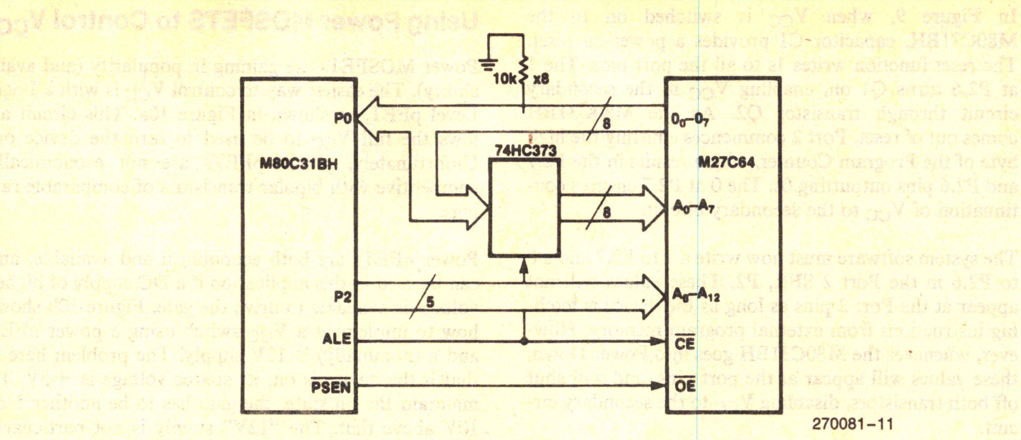


Figure 11a. M80C31BH + M27C64

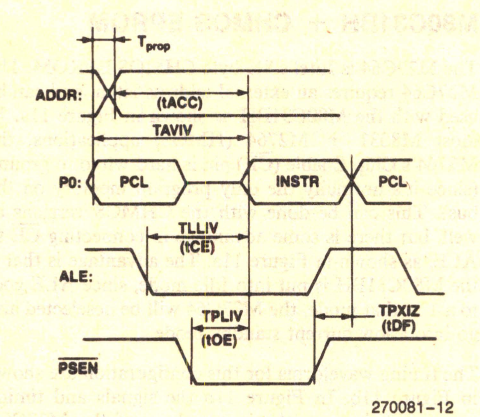
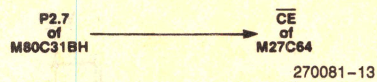


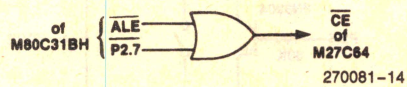
Figure 11b. Timing Waveforms for M80C31BH + M27C64

select the EPROM when the CPU is in Power Down. If Idle is never invoked, $\overline{CE}$ of the EPROM can be connected to P2.7 of the M80C31BH, as shown in Figure 12a. In normal operation, P2.7 will be emitting the MSB of the Program Counter, which is 0 if the program contains less than 32K of code. Then when the CPU goes into Power Down, the Port 2 pins emit P2 SFR data, which puts a 1 at P2.7, thus deselecting the EPROM.

If Idle and Power Down are both going to be used, $\overline{CE}$ of the EPROM can be driven by the logical OR of ALE and P2.7, as shown in Figure 12b. In Idle, ALE = 1 will deselect the EPROM, and in Power Down, P2.7 = 1 will deselect it.



a. Power Down Is Used But Not Idle



b. Idle and Power Down Both Used

Figure 12. Modifications to M80C31BH/M27C64 Interface

Pulldown resistors are shown in Figure 11 under the assumption that something on the bus is going to have its V_{CC} removed during Power Down. If this is not the case, pullups can be used as well as pulldowns.

SCANNING A KEYBOARD

There are many different kinds of keyboards, but alphanumeric keyboards generally consist of a matrix of 8 scan lines and 8 receive lines as shown in Figure 13. Each set of lines connects to one port of the microcontroller. The software has written 0s to the scan lines, and 1s to the receive lines. Pressing a key connects a scan line to a receive line, thus pulling the receive line to a logic low.

The 8 receive lines are ANDed to one of the external interrupt pins, so that pulling any of the receive lines low generates an interrupt. The interrupt service routine has to identify the pressed key, if only one key is down, and convert that information to some useful output. If more than one key in the line matrix is found to be pressed, no action is taken. (This is a "two key lock-out" scheme.)

On some keyboards, certain keys (Shift, Control, Escape, etc.) are not a part of the line matrix. These keys would connect directly to a port pin on the microcontroller, and would not cause lock-out if pressed simultaneously with a matrix key, nor generate an interrupt if pressed singly.

Normally the microcontroller would be in Idle mode when a key has not been pressed, and another task is not in progress. Pressing a matrix key generates an interrupt, which terminates the Idle. The interrupt service routine would first call a 30 ms (or so) delay to debounce the key, and then set about the task of identifying which key is down.

First, the current state of the receive lines is latched into an internal register. If a single key is down, all but one of these lines would be read as 1s. Then 0s are written to the receive lines and 1s to the scan

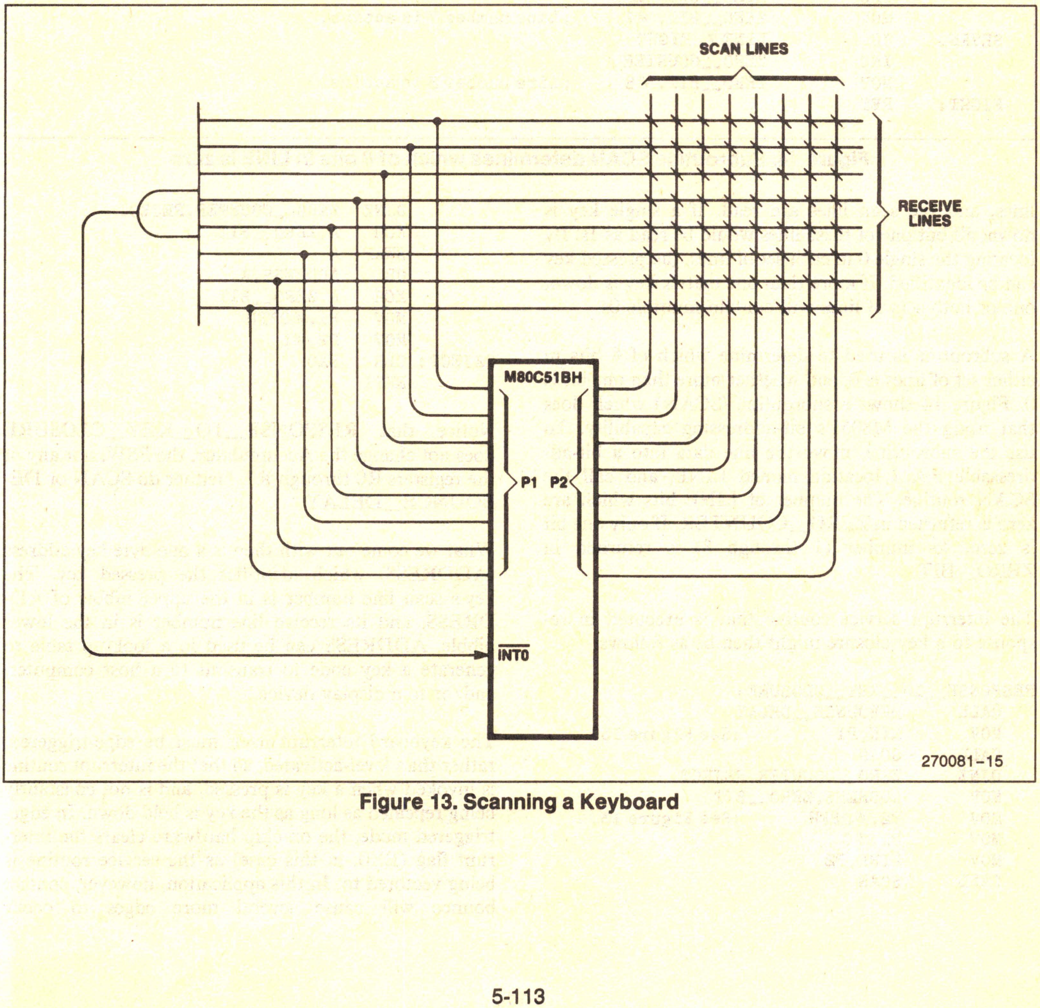


Figure 13. Scanning a Keyboard

```
SCAN:    MOV    ZERO__COUNTER, #0  ;ZERO__COUNTER counts the number of 0s in LINE.
         JB     Line.0, ONE        ;Test Line bit 0.
         INC    ZERO__COUNTER      ;If LINE.0 = 0, increment ZERO__COUNTER
         MOV    ZERO__BIT, #1      ;   and record that line number 1 is active.
ONE:     JB     LINE.1, TWO        ;Procedure continues for other LINE bits.
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #2      ;Line number 2 is active.
TWO:     JB     LINE.2, THREE
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #3      ;Line number 3 is active.
THREE:   JB     LINE.3, FOUR
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #4      ;Line number 4 is active.
FOUR:    JB     LINE4, FIVE
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #5      ;Line number 5 is active.
FIVE:    JB     LINE5, SIX
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #6      ;Line number 6 is active.
SIX:     JB     LINE6, SEVEN
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #7      ;Line number 7 is active.
SEVEN:   JB     LINE7, EIGHT
         INC    ZERO__COUNTER
         MOV    ZERO__BIT, #8      ;Line number 8 is active.
EIGHT:   RET
```

Figure 14. Subroutine SCAN determines which of 8 bits in LINE is zero

lines, and the scan lines are read. If a single key is down, all but one of these lines would be read as 1s. By locating the single 0 in each set of lines, the pressed key can be identified. If more than one matrix key is down, one or both sets of lines will contain multiple 0s.

A subroutine is used to determine which of 8 bits in either set of lines is 0, and whether more than one bit is 0. Figure 14 shows a subroutine (SCAN) which does that using the M8051's bit-addressing capability. To use the subroutine, move the line data into a bit-addressable RAM location named LINE, and call the SCAN routine. The number of LINE bits which are zero is returned in ZERO__COUNTER. If only one bit is zero, its number (1 through 8) is returned in ZERO__BIT.

The interrupt service routine that is executed in response to a key closure might then be as follows:

```
RESPONSE__TO__KEY__CLOSURE:
   CALL     DEBOUNCE__DELAY
   MOV      LIN,P1            ;See Figure 13.
   CALL     SCAN
   DJNZ     ZERO__COUNTER,REJECT
   MOV      ADDRESS,ZERO__BIT
   MOV      P2,#0FFH          ;See Figure 13.
   MOV      P1,#0
   MOV      LINE,P2
   CALL     SCAN
   DJNZ     ZERO__COUNTER,REJECT
   XCH      A,ZERO__BIT
   SWAP     A
   ORL      ADDRESS,A
   XCH      A,ZERO__BIT
   MOV      P1,#0FFH
   MOV      P2,#0
REJECT: CLR EX0
   RETI
```

Notice that RESPONSE__TO__KEY__CLOSURE does not change the Accumulator, the PSW, nor any of the registers R0 through R7. Neither do SCAN or DEBOUNCE__DELAY.

What we come out with then is a one-byte key address (ADDRESS) which identifies the pressed key. The key's scan line number is in the upper nibble of ADDRESS, and its receive line number is in the lower nibble. ADDRESS can be used in a look-up table to generate a key code to transmit to a host computer, and/or to a display device.

The keyboard interrupt itself must be edge-triggered, rather than level-activated, so that the interrupt routine is invoked when a key is pressed, and is not constantly being repeated as long as the key is held down. In edge-triggered mode, the on-chip hardware clears the interrupt flag (EX0, in this case) as the service routine is being vectored to. In this application, however, contact bounce will cause several more edges to occur

after the service routine has been vectored to, during the DEBOUNCE__DELAY routine. Consequently it is necessary to clear EX0 again in software before executing RETI.

The debounce delay routine also takes advantage of the Idle mode. In this routine a timer must be preloaded with a value appropriate to the desired length of delay. This value would be

$$\text{timer preload} = \frac{(\text{osc KHz}) \times (\text{delay time ms})}{12}$$

For example, with a 3.58 MHz oscillator frequency, a 30 ms delay could be obtained using a preload value of −8950, or DD0A, in hex digits.

In the debounce delay routine (Figure 15), the timer interrupt is enabled and set to a higher priority than the keyboard interrupt, because as we invoke Idle, the keyboard interrupt is still "in progress". An interrupt of the same priority will not be acknowledged, and will not terminate the Idle mode. With the timer interrupt set to priority 1, while the keyboard interrupt is at priority 0, the timer interrupt, when it occurs, will be acknowledged and will wake up the CPU. The timer interrupt service routine does not itself have to do anything. The service routine might be nothing more than a single RETI instruction. RETI from the timer interrupt service routine then returns execution to the debounce delay routine, which shuts down the timer and returns execution to the keyboard service routine. An appropriate debounce delay routine is shown in Figure 15.

DRIVING AN LCD

AN LCD (Liquid Crystal Display) consists of a backplane and any number of segments or dots which will be used to form the image being displayed. Applying a voltage (nominally 4 or 5V) between any segment and the backplane causes the segment to darken.

The only catch is that the polarity of the applied voltage has to be periodically reversed, or else a chemical reaction takes place in the LCD which causes deterioration and eventual failure of the liquid crystal.

To prevent this happening, the backplane and all the segments are driven with an AC signal, which is derived from a rectangular voltage waveform. If a segment is to be "off" it is driven by the same waveform as the backplane. Thus it is always at backplane potential. If the segment is to be "on" it is driven with a waveform that is the inverse of the backplane waveform. Thus it has about 5V of periodically changing polarity between it and the backplane.

With a little software overhead, the M80C51BH can perform this task without the need for additional LCD drivers. The only drawback is that each LCD segment uses up one port pin, and the backplane uses one more. If more than, say, two 7-segment digits are being driven, there aren't many port pins left for other tasks. Nevertheless, assuming a given application leaves enough port pins available to support this task, the considerations for driving the LCD are as follows.

Suppose, for example, it is a 2-digit display with a decimal point. One port (TENS__DIGIT) connects to the 7 segments of the tens digit plus the backplane. Another port (ONES__DIGIT) connects to a decimal point plus the 7 segments of the ones digit.

One of the M80C51BH's timers is used to mark off half-periods of the drive voltage waveform. The LCD drive waveform should have a rep rate between 30 and 100 Hz, but it's not very critical. A half-period of 12 ms will set the rep rate to about 42 Hz. The preload/reload value to get 12 ms to rollover is the 2's complement negative of the oscillator frequency in KHz: If the oscillator frequency is 3.58 MHz, the reload value is −3580, or F204 in hex digits.

```
DEBOUNCE__DELAY:
          MOV       TL1, #TL1__PRELOAD ; Preload low byte.
          MOV       TH1, #TH1__PRELOAD ; Preload high byte.
          SETB      ET1                ; Enable Timer 1 interrupt.
          SETB      PT1                ; Set Timer 1 interrupt to high priority.
          SETB      TR1                ; Start timer running.
          ORL       PCON, #1           ; Invoke Idle mode.
;
; The next instruction will not be executed until the delay times out.
;
          CLR       TR1                ; Stop the timer.
          CLR       PT1                ; Back to Priority 0 (if desired).
          CLR       ET1                ; Disable Timer 1 interupt (if desired).
          RET                          ; Continue keyboard scan.
```

Figure 15. Subroutine DEBOUNCE__DELAY puts the M80C51BH into Idle during the delay time

Now, the M80C51BH would normally be in Idle, to conserve power, during the time that the LCD and other tasks are not requiring servicing. When the timer rolls over it generates an interrupt, which brings the M80C51BH out of Idle. The service routine reloads the timer (for the next rollover), and inverts the logic levels of all the pins that are connected to the LCD. It might look like this:

```
LCD__DRIVE__INTERRUPT:
  MOV      TL1,#LOW(--XTAL__FREQ)
  MOV      TH1,#HIGH(--XTAL__FREQ)
  XRL      TENS__DIGIT,#0FFH
  XRL      ONES__DIGIT,#0FFH
  RETI
```

To update the display, one would use a look-up table to generate the characters. In the table, "on" segments are represented as 1s, and "off" segments as 0s. The backplane bit is represented as a 0. The quantity to be displayed is stored in RAM as a BCD value. The look-up table operates on the low nibble of the BCD value, and produces the bit pattern that is to be written to either the ones digit or the tens digit. Before the new patterns can be written to the LCD, the LCD drive interrupt has to be disabled. That is to prevent a polarity reversal from taking place between the times the two digits are written. An update subroutine is shown in Figure 16.

RESONANT TRANSDUCERS

Analog transducers are often used to convert the value of a physical property, such as temperature, pressure, etc., to an analog voltage. These kinds of transducers then require an analog-to-digital converter to put the measurement into a form that is compatible with a digital control system. Another kind of transducer is now becoming available that encodes the value of the physical property into a signal that can be directly read by a digital control system. These devices are called resonant transducers.

Resonant transducers are oscillators whose frequency depends in a known way on the physical property being measured. These devices output a train of rectangular pulses whose repetition rate encodes the value of quantity being measured. The pulses can in most cases be fed directly into the M80C51BH, which then measures either the frequency or period of the incoming signal, basing the measurement on the accuracy of its own clock oscillator. The M80C51BH can even do this in its sleep; that is, in Idle.

When the frequency of period measurement is completed, the C51BH wakes itself up for a very short time to perform a sanity check on the measurement and convert it in software to any scaling of the measured quantity that may be desired. The software conversion can include corrections for nonlinearities in the transducer's transfer function.

Resolution is also controlled by software, and can even be dynamically varied to meet changing needs as a situation becomes more critical. For example, in a process controller you can increase your resolution ("fine tune" the control, as it were) as the process approaches its target.

The nominal reference frequency of the output signal from these devices is in the range of 20 Hz to 500 KHz, depending on the design. Transducers are available

```
UPDATE__LCD:
        CLR     ET1                     ;Disable LCD drive interrupt.
        MOV     DPTR,#TABLE__ADDRESS    ;Look-up table begins at TABLE__ADDRESS
        MOV     A,BCD__VALUE            ;Digits to be displayed.
        SWAP    A                       ;Move tens digit to low nibble.
        ANL     A,#0FH                  ;Mask off high nibble.
        MOVC    A,@A+DPTR               ;Tens digit pattern to accumulator.
        MOV     TENS__DIGIT,A           ;Update LCD tens digit.
        MOV     A,BCD__VALUE            ;Digits to be displayed.
        ANL     A,#0FH                  ;Mask off tens digit.
        MOVC    A,@A+DPTR               ;Ones digit pattern to accumulator.
        MOV     C,DECIMAL__POINT        ;Add decimal point to segment
        MOV     ACC.7,C                 ;   pattern.   Update LCD decimal point
        MOV     ONES__DIGIT,A           ;   and ones digit.
        SETB    ET1                     ;Re-enable LCD drive interrupt.
        RET
```

Figure 16. UPDATE__LCD routine writes two digits to an LCD

that have a full scale frequency shift of 2 to 1. The transducer operates from a supply voltage range of 3V to 20V, which means it can operate from the same supply voltage as the M80C51BH. At 5V, the transducer draws less the 5 mA (reference 7). It can normally be connected directly to one of the C51BH's port pins, as shown in Figure 21.

Frequency Measurements

Measuring a frequency means counting pulses for a known sample time. Two timer/counters can be used, one to mark off the sample time and one to count pulses. If the frequency being counted doesn't exceed 50 KHz or so, one may equally well connect the transducer signal to one of the external interrupt pins, and count pulses in software. That frees up one timer, with very little cost in CPU time.

The count that is directly obtained is T × F, where T is the sample time and F is the frequency. The full scale range is T × (Fmax-Fmin). For n-bit resolution

$$1\ \text{LSB} = \frac{T \times (\text{Fmax-Fmin})}{2^n}.$$

Therefore the sample time required for n-bit resolution is

$$T = \frac{2^n}{\text{Fmax-Fmin}}.$$

For example, 8-bit resolution in the measurement of a frequency that varies between 7 KHz and 9 KHz would require, according to this formula, a sample time of 128 ms. The maximum acceptable frequency count would be 128 ms × 9 KHz = 1152 counts. The minimum would be 896 counts. Subtracting 896 from each frequency count (or presetting the frequency counter to −896 = 0FC80H) would allow the frequency to be reported on a scale of 0 to FF in hex digits.

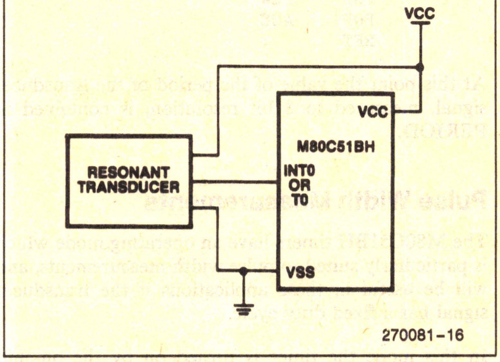


Figure 17. Resonant Transducer does not require an A/D converter.

To implement the measurement, one timer is used to establish the sample time. The timer is preset to a value that causes it to roll over at the end of the sample time, generating an interrupt and waking the CPU from its Idle mode. The required preset value is the 2's complement negative of the sample time measured in machine cycles. The conversion from sample time to machine cycles is to multiply it by 1/12 the clock frequency. For example, if the clock frequency is 12 MHz, then a sample time of 128 ms is

$$(128\ \text{ms}) \times (12000\ \text{KHz})/12 = 128000\ \text{machine cycles}.$$

Then the required preset value to cause the timer to roll over in 128 ms is

$$-128000 = \text{FE0C00, in hex digits.}$$

Note that the preset value is 3 bytes wide, whereas the timer is only 2 bytes wide. This means the timer must be augmented in software in the timer interrupt routine to three bytes. The M80C51BH has a DJNZ instruction (decrement and jump if not zero) that makes it easier to code the third timer byte to count down instead of up. If the third timer byte counts down, its reload value is the 2's complement of what it would be for an up-counter. For example, if the 2's complement of the sample time is FE0C00, then the reload value for the third timer byte would be 02, instead of FE. The timer interrupt routine might then be:

```
TIMER__INTERRUPT__ROUTINE:
            DJNZ        THIRD__TIMER__BYTE,OUT
            MOV         TLO,#0
            MOV         THO,#0CH
            MOV         THIRD__TIMER__BYTE,#2
            MOV         FREQUENCY,COUNTER__LO
;Preset COUNTER to       -896:
            MOV         COUNTER__LO,#80H
            MOV         COUNTER__HI,#0FCH
OUT:        RETI
```

At this point the value of the frequency of the transducer signal, measured to 8 bit resolution, is contained in FREQUENCY. Note that the timer can be reloaded on the fly. Note too that for 8-bit resolution only the low byte of the frequency counter needs to be read, since the high byte is necessarily 0. However, one may want to test the high byte to ensure that it is zero, as a sanity check on the data. Both bytes, of course must be reloaded.

Period Measurements

Measuring the period of the transducer signal means measuring the total elapsed time over a known number, N, of transducer pulses. The quantity that is di-

rectly measured is NT, where T is the period of the transducer signal in *machine cycles.* The relationship between T in machine cycles and the transducer frequency F in arbitrary frequency units is

$$T = \frac{Fxtal}{F} \times (1/12),$$

where Fxtal is the M80C51BH clock frequency, in the same units as F.

The full scale range then is Nx(Tmax-Tmin). For n-bit resolution

$$1\ LSB = \frac{Nx(Tmax\text{-}Tmin)}{2^n}.$$

Therefore the number of periods over which the elapsed time should be measured is

$$N = \frac{2^n}{Tmax\text{-}Tmin}.$$

However, N must also be an integer. It is logical to evaluate the above formula (don't forget Tmax and Tmin have to be in machine cycles) and select for N the next higher integer. This selection gives a period measurement that has somewhat more than n-bit resolution, but it can be scaled back if desired.

For example, suppose we want 8-bit resolution in the measurement of the period of a signal whose frequency varies from 7.1 KHz to 9 KHz. If the clock frequency is 12 MHz, then Tmax is (12000 KHz/7.1 KHz) × (1/12) = 141 machine cycles. Tmin is 111 machine cycles. The required value for N, then, is 256/(141-111) = 8.53 periods, according to the formula. Using N = 9 periods will give a maximum NT value of 141 × 9 = 1269 machine cycles. The minimum NT will be 111 × 9 = 999 machine cycles. A lookup table can be used to scale these values back to a range of 0 to 255, giving precisely the 8-bit resolution desired.

To implement the measurement, one timer is used to measure the elapsed time, NT. The transducer is connected to one of the external interrupt pins, and this interrupt is configured to the transition-activated mode. In the transition-activated mode every 1-to-0 transition in the transducer output will generate an interrupt. The interrupt routine counts transducer pulses, and when it gets to the predetermined N, it reads and clears the timer. For the specific example cited above, the interrupt routine might be:

```
INTERRUPT__RESPONSE:
          DJNZ     N,OUT
          MOV      N,#9
          CLR      EA
          CLR      TR1
          MOV      NT__LO,TL1
          MOV      NT__H1,TH1
          MOV      TL1,#9
          MOV      TH1,#0
          SETB     TR1
          SETB     EA
          CALL     LOOKUP__TABLE
OUT:      RETI
```

In this routine a pulse counter N is decremented from its preset value, 9, to zero. When the counter gets to zero it is reloaded to 9. Then all interrupts are blocked for a short time while the timer is read and cleared. The timer is stopped during the read and clear operations, so "clearing" it actually means presetting it to 9, to make up for the 9 machine cycles that are missed while the timer is stopped.

The subroutine LOOKUP__TABLE is used to scale the measurement back to the desired 8-bit resolution. It can also include built-in corrections for errors or nonlinearities in the transducer's transfer function.

The subroutine uses the MOVC A, @A+DPTR instruction to access the table, which contains 270 entries commencing at the 16-bit address referred to as TABLE. The subroutine must compute the address of the table entry that corresponds to the measured value of NT. This address is

```
DPTR = TABLE + NT -NTMIN,
```

where NTMIN = 999, in this specific example.

```
LOOKUP__TABLE:
          PUSH     ACC
          PUSH     PSW
          MOV      A,#LOW(TABLE-NTMIN)
          ADD      A,NT__LO
          MOV      DPL,A
          MOV      A,#HIGH(TABLE-NTMIN)
          ADDC     A,NT__HI
          MOV      DPH,A
          CLR      A
          MOVC     A,@A +DPTR
          MOV      PERIOD,A
          POP      PSW
          POP      ACC
          RET
```

At this point the value of the period of the transducer signal, measured to 8 bit resolution, is contained in PERIOD.

Pulse Width Measurements

The M80C51BH timers have an operating mode which is particularly suited to pulse width measurements, and will be useful in these applications if the transducer signal has a fixed duty cycle.

In this mode the timer is turned on by the on-chip circuitry in response to an input high at the external interrupt pin, and off by an input low, and it can do this while the M80C51BH is in Idle. (The "GATE" mode

of timer operation is described in the Intel Microcontroller Handbook.) The external interrupt itself can be enabled, so the same 1-to-0 transition from the transducer that turns off the timer also generates an interrupt. The interrupt routine then reads and resets the timer.

The advantage of this method is that the transducer signal has direct access to the timer gate, with the result that variations in interrupt response time have no effect on the measurement.

Resonant transducers that are designed to fully exploit the GATE mode have an internal divide-by-N circuit that fixes the duty cycle at 50% and lowers the output frequency to the range of 250 to 500 Hz (to control RFI). The transfer function between transducer period and measurand value is approximately linear, with known and repeatable error functions.

HMOS/CHMOS INTERCHANGEABILITY

The CHMOS version of the M8051 is architecturally identical with the HMOS version, but there are nevertheless some important differences between them which the designer should be aware of. In addition, some applications require interchangeability between HMOS and CHMOS parts. The differences that need to be considered are as follows:

External Clock Drive

To drive the HMOS M8051 with an external clock signal, one normally grounds the XTAL1 pin and drives the XTAL2 pin. To drive the CHMOS M8051 with an external clock signal, one must drive the XTAL1 pin and leave the XTAL2 pin unconnected. The reason for the difference is that in the HMOS M8051, it is the XTAL2 pin that drives the internal clocking circuits, whereas in the CHMOS version it is the XTAL1 pin that drives the internal clocking circuits.

There are several ways to design an external clock drive to work with both types. For low clock frequencies (below 6 MHz), the HMOS M8051 can be driven the same way as the CHMOS version, namely, through XTAL1 with XTAL2 unconnected. Another way is to drive both XTAL1 **and** XTAL2; that is, drive XTAL1 and use and external inverter to derive from XTAL1 a signal with which to drive XTAL2.

In either case, a 74HC or 74HCT circuit makes an excellent driver for XTAL1 and/or XTAL2, because neither the HMOS nor the CHMOS XTAL pins have TTL-like input logic levels.

Unused Pins

Unused pins of Ports 1, 2, and 3 can be ignored in both HMOS and CHMOS designs. The internal pullups will put them into a defined state. Unused Port 0 pins in M8051 applications can be ignored, even if they're floating. But in M80C51BH applications, these pins should not be left afloat. They can be externally pulled up or down, or they can be internally pulled down by writing 0s to them.

M8031/M80C31BH designs may or may not need pullups on Port 0. Pullups aren't needed for program fetches, because in bus operations the pins are actively pulled high or low by either the M8031 or the external program memory. But they are needed for the CHMOS part if the Idle or Power Down mode is invoked, because in these modes Port 0 floats.

Logic Levels

If V_{CC} is between 4.5V and 5.5V, an input signal that meets the HMOS M8051's input logic levels will also meet the CHMOS M80C51BH's input logic levels (except for XTAL1/XTAL2 and RST). For the same V_{CC} condition, the CHMOS device will reach or surpass the output logic levels of the HMOS device. The HMOS device will not necessarily reach the output logic levels of the CHMOS device. This is an important consideration if HMOS/CHMOS interchangeability must be maintained in an otherwise CMOS system.

HMOS M8051 outputs that have internal pullups (Ports 1, 2, and 3) "typically" reach 4V or more if I_{OH} is zero, but not fast enough to meet timing specs. Adding an external pullup resistor will ensure the logic level, but still not the timing, as shown in Figure 18. If timing is an issue, the best way to interface HMOS or CMOS is through a 74HCT circuit.

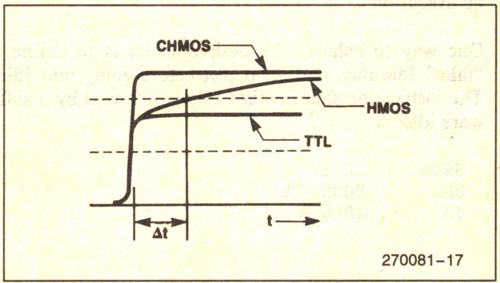


Figure 18. 0-to-1 Transition shows unspec'd delay (Δt) in HMOS to 74HC Logic

Idle and Power Down

The Idle and Power Down modes exist only on the CHMOS devices, but if one wishes to preserve the capability of interchanging HMOS and CHMOS M8051s, the software has to be designed so that the HMOS parts will respond in an acceptable manner when a CHMOS reduced power mode is invoked.

For example, an instruction that invokes Power Down can be followed by a "JMP $":

```
ORL     PCON,#2
JMP     $
```

The CHMOS and HMOS parts will respond to this sequence of code differently. The CHMOS part, going into a normal CHMOS Power Down Mode, will stop fetching instructions until it gets a hardware reset. The HMOS part will go through the motions of executing the ORL instruction, and then fetch the JMP instruction. It will continue fetching and executing JMP $ until hardware reset.

Maintaining HMOS/CHMOS M8051 interchangeability in response to Idle requires more planning. The HMOS part will not respond to the instruction that puts the CHMOS part into Idle, so that instruction needs to be followed by a software idle. This would be an idling loop which would be terminated by the same conditions that would terminate the CHMOS's hardware Idle. Then when the CHMOS device goes into Idle, the HMOS version executes the idling loop, until either a hardware reset or an enabled interrupt is received. Now if Idle is terminated by an interrupt, execution for the CHMOS device will proceed after RETI from the instruction following the one that invoked Idle. The instruction following the one that invoked Idle is the idling loop that was inserted for the HMOS device. At this point, both the HMOS and CHMOS devices must be able to fall through the loop to continue execution.

One way to achieve the desired effect is to define a "fake" Idle flag, and set it just before going into Idle. The instruction that invokes Idle is followed by a software idle:

```
SETB    IDLE
ORL     PCON,#1
JB      IDLE,$
```

Now the interrupt that terminates the CHMOS's Idle must also break the software idle. It does so by clearing the "Idle" bit:

```
CLR     IDLE
RETI
```

Note too that the PCON register in the HMOS M8051 contains only one bit, SMOD, whereas the PCON register in CHMOS contains SMOD plus four other bits. Two of those other bits are general purpose flags. Maintaining HMOS/CHMOS interchangeability requires that these flags not be used.

REFERENCES

1. Pawloski, Moroyan, Altnether, "Inside CMOS Technology," *BYTE magazine,* Sept. 1983. Available as Article Reprint AR-302.

2. Kokkonen, Pashley, "Modular Approach to C-MOS Technology Tailors Process to Application," *Electronics,* May, 1984. Available as Article Reprint AR-332.

3. Williamson, T., *Designing Microcontroller Systems for Electrically Noisy Environments,* Intel Application Note AP-125, Feb. 1982.

4. Williamson, T., "PC Layout Techniques for Minimizing Noise," *Mini-Micro Southeast,* Session 9, Jan. 1984.

5. Altnether, J., *High Speed Memory System Design Using 2147H,* Intel Application Note AP-74, March 1980.

6. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 292–297, Aug. 1981.

7. *Digital Sensors by Technar,* Technar Inc., 205 North 2nd Ave., Arcadia, CA 91006.

8. Geisert, C. & Marks, K., "New Generation MOS Chips Display Radiation-Hardened Properties," *Military/ Space Electronics Design,* February 1985. Available as Article Reprint AR-382.

intel®

APPLICATION NOTE

AP-282

September 1986

29C53 Transceiver Line Interfacing

JAGTINDER S. BOLARIA
TELECOM PRODUCT MARKETING

Order Number: 270209-001

INTRODUCTION

Presently, the majority of the transmission from the telephone to the Central Switching system is analog. For this purpose the circuitry interfacing to the twisted pair line is optimized to operate between 300 and 3400 Hz. The essential line interface functions consist of isolation, over voltage protection, signaling, power feeding and a ringing signal insertion. With the advent of ISDN (Integrated Services Digital Network) these functions have to be reassessed.

ISDN is implemented with digital transmission from the subscriber to the switch, which in turn offers the user various data services in addition to the voice service. CCITT has various recommendations for the implementation of the ISDN network. Of these, I.430 details the basic rate access i.e. the physical communications between a terminal and the first level of switching. For I.430, Intel offers a transceiver which is capable of operating at either end of the loop, namely the 29C53.

The 29C53 is a four wire (two for transmit and two for receive) transceiver operating over the "S" loop. The data transmitted by the 29C53 at the switch and the terminal is at a rate of 192 kb/s; the effective data throughput is 144 kb/s. This data consists of two bearer channels of 64 kb/s each (B1 + B2) and a 16 kb/s D channel. The 29C53, additionally, incorporates some protocol processing for the D channel. This transceiver has four interfaces, namely the microprocessor port, a general purpose I/O port, the SLD port and the "S" loop interface. It is the loop interface requirements that are addressed by this application note.

This note will analyze the line interface requirement at both the line card and the terminal, and will offer general implementations. These implementations will address power feeding, the protection circuitry, the line transformers and power extraction. Throughout this brief, the approach has been to present various alternate concepts which may assist the designer in addressing a specific application.

LINE INTERFACE

Both at the line card and the terminal, there is a need to provide isolation for the circuitry from the line itself. As well as isolation, it is also necessary to protect the equipment from any overvoltage conditions on the line. Additionally the system may be designed to provide phantom power feeding i.e. the switching system delivers power to the terminal over the "S" loop. Unlike its analog counter part the digital line card does not need to send a ringing signal owing to the fact that all signaling is accommodated via the D channel.

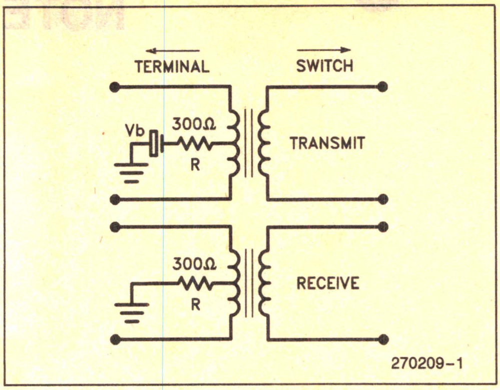


Figure 1. Voltage Feeding

POWER FEEDING

Figure 1 shows the CCITT recommended technique of phantom power feeding as described in section 9 of I.430. The current splits evenly between the two secondary windings. This in turn produces equal and opposite fluxes in the transformer, that cancel each other out, thus preventing the core from saturating. The equality of the fluxes in the secondary will depend on the longitudinal balance of the transformer and the transmission line.

The scheme shown on Figure 1 may be wasteful of power when feeding short lines. One way around this would be to have a constant current feed, which will make the power consumption independent of the length of line. Figure 2 shows such an implementation.

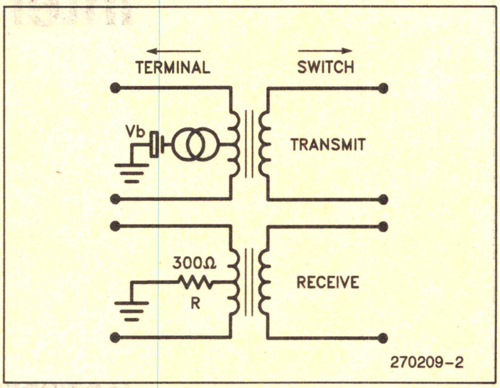


Figure 2. Current Feeding

One way of reducing the power dissipation over the loop is to provide a variable voltage source, instead of the traditional fixed voltage. This can be accomplished by using a DC to DC converter, or a switching regulator. The feedback circuit of the switching regulator can be used to ensure that the regulator provides just enough voltage to maintain a pre-defined feed current down any length of line. The DC to DC converter can have a built in threshold detector, which would be used to release the line in case excessive currents are being drawn.

In the event of mains power loss, it is often required to maintain a minimal voice service powered off the line. Figure 3 shows the block diagram of a digital telephone, illustrating the necessary components required to maintain a voice service.

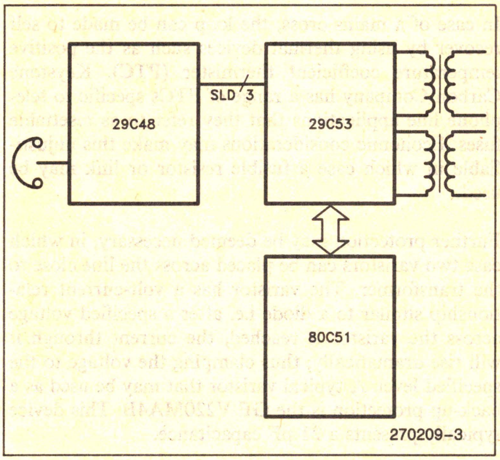


Figure 3. Digital Telephone

The 80C51 is a low power microcontroller while the 29C48 is an SLD compatible combo (codec and filter). The gains through the 29C48 can be set externally or programmed by the microcontroller via the SLD interface. The 29C48 is designed to allow insertion of sidetone and DTMF (dual tone multi-frequency); both these features are presently used to provide feedback to the user. The power required to provide a voice service can be split as shown below:

29C48	90 mW
29C53	100 mW
80C51	120 mW
Ancillary components	80 mW
TOTAL	400 mW maximum

Assuming a 70% efficient power converter, then 570 mW of power is required by the terminal. If the lowest allowed voltage drop at the terminal is 15V then the required current is 38 mA. Assuming, in accordance with I.430, that the minimum power feed voltage at the source is 38V, then a loop resistance of 600Ω can be tolerated. This is ample for 1 km distance of the transmission lines presently in use (i.e. 1500m of 26AWG presents 400Ω at 20°C).

PROTECTION

Next, let us examine the question of protection. A telecommunication system comprises subscribers linked together through the cable plant and a switching network. The cable plant consists of multiple pairs of transmission lines, either suspended on poles, or buried in the earth. In either case, transient energy can be coupled from lightning (or other electromagnetic events) and conducted to the switch or the terminal. The other major source of transient energy is the commercial AC power system, where high currents that accompany faults can induce overvoltage in the lines, or the power lines can fall and make contact with the telephone lines. The latter is sometimes referred to as a mains or power cross.

It is generally agreed, as shown in Figure 4, that two or more levels of protection are required. The primary protector is usually placed on the line at a distance greater than 25m from the line card. The impedance of the line will ensure that the primary protector will operate first and the secondary protector will not be exposed to the full surge. If the primary protector is to be placed closer to the secondary, then a small resistor can be inserted in series with the line between the primary and the secondary protector (1). A 5Ω 3W resistor or a positive temperature coefficient resistor may be used. During a surge, the voltage drop across the resistor will increase allowing the voltage across the primary protector to build up thus driving it to conduction.

The primary protection can be a gas discharge tube, such as the General Instrument three terminal PMT3-(310). These devices consist of spaced metallic gaps enclosed in a combination of gases at low pressure. In the event of a surge, the gap breaks down, diverting the transient and thus rerouting the energy. These devices can be operated a number of times and present a capacitance of less than 5 pF. Since the templates in Figures 10 and 11 of I.430 specify a low output capacitance for the terminal and the network terminator, the low output capacitance feature of the gas discharge tube makes it ideal for ISDN i.e. it will have a minimal effect on the line drivers.

The secondary protection can be provided by Schottky diodes chosen for the low voltage drop and capacitance across them. The diodes are placed between the power supplies and the loop interface pins on the 29C53, thus forming a diode bridge across the line. This will ensure that the voltage on these pins does not exceed the power supplies by more than approximately 300 mV, thus fulfilling the specification that the voltage on any pin

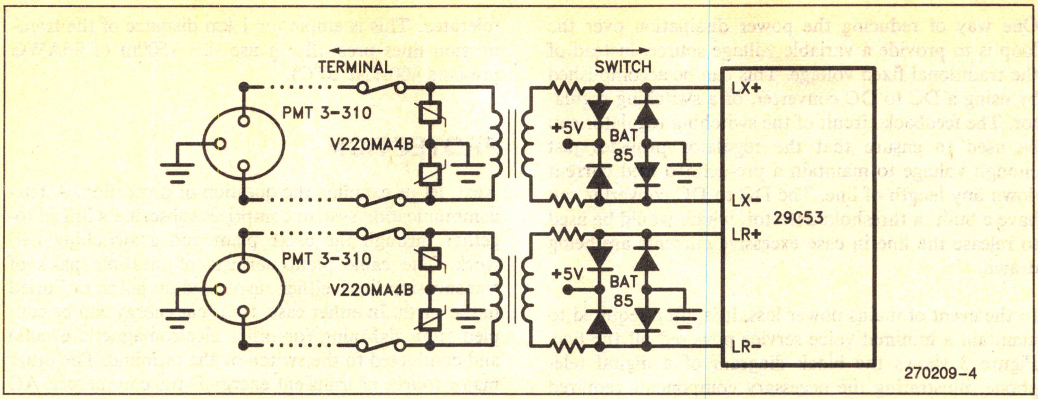


Figure 4. Protection

may not exceed the power supply by more than 500 mV. The 5V and ground connections to the diodes should be as close as possible to the 29C53 power supply pins, which in turn should be decoupled by a 0.1 μF capacitor. The capacitor serves a secondary function of bypassing surge currents. The particular diodes chosen are dependent on the expected surge current, however, BAT85 from Philips used in this application can withstand 200 mA forward current while presenting a maximum of 10 pF capacitance across it. It may be desired to limit the maximum current through the diodes. This can be achieved by placing a small value resistor in series with the diodes and the transformer. The value of this resistance can be extracted from the transformer design discussion. For the receive direction it is possible to replace the diode bridge by placing a resistance in series with the 29C53 receive pins. This series resistance will limit the surge current that the 29C53 is exposed to. The value of this resistance is limited by the input impedance presented by the 29C53 and the loss that can be tolerated in the received signal. The receive differential input impedance of the 29C53 is 100 KΩ, hence a 10 KΩ resistor in each arm will reduce the received signal by 17%.

In case of a mains cross, the loop can be made to self recover by using thermal devices such as the positive temperature coefficient thermister (PTC). Keystone Carbon Company has a range of PTCs specific to telephone line applications that they refer to as resettable fuses. Economic considerations may make this unjustifiable in which case a fusible resistor or link may be used.

Further protection may be deemed necessary, in which case two varistors can be placed across the line close to the transformer. The varistor has a volt-current relationship similar to a diode i.e. after a specified voltage across the varistor is reached, the current through it will rise dramatically; thus clamping the voltage to the specified level. A typical varistor that may be used as a back-up protection is the GE V220MA4B. This device typically presents a 21 pF capacitance.

The ideas discussed thus far are encompassed in Figure 5 for a minimal component count protection scheme.

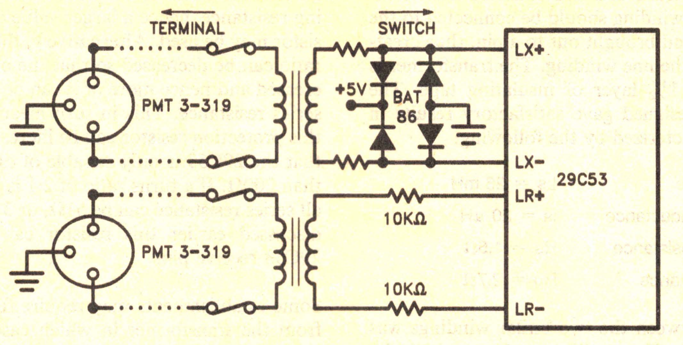


Figure 5. Protection with Minimal Components

LINE TRANSFORMER

A transformer is used at both the terminal and the line card to provide isolation from the line. A well balanced I.430 transformer resolves the issue of DC currents since they induce self-cancelling fluxes. Generally speaking, a pulse transformer with minimum leakage inductance and self capacitance is required. The impedance templates in I.430 specify the minimum value of the inductance required at the line side. This value can be calculated to be 20 mH. The transformer used with the 29C53 needs to be a 2.5:1 ratio from the 29C53. A further requirement is to minimize the winding resistance, so that a minimal voltage is dropped across it. The transformer design discussed below can be used with the 29C53 at either the line card or the terminal. Alternatively it can be used for example purposes to aid designs.

The RM series of ferrite cores are chosen to facilitate easy winding and PCB mounting, additionally the RM series is available internationally from various vendors–Ferroxcube in the U.S. and Mullard in Europe, to name two. The RM6 core was selected to be the smallest size that accommodates wiring which does not exceed the maximum allowable DC resistance. The core material has to have a high enough permeability to allow the 20 mH inductance with a minimum number of turns hence, the Ferroxcube core material 3E2A was selected. This material has a very high inductance factor, A_L. This is given by the manufacturer as the inductance (in mH) per 1000 turns.

For the core RM6PL00-3E2A

$A_L = 6710 \pm 25\%$

Therefore minimum

$A_L = 5032 \cong 5000$

The number of turns, Ns, required for 20 mH is given by:

$Ns = 10^3 \sqrt{L/A_L}$ L - required inductance in mH

$Ns = 70$ turns - assume 25 mH is required

The 29C53 side winding will require 2.5 times this number of turns.

$Np = 175$ turns

The transformer is now ready to be wound, the 32 gauge wire will just fill the RM6PCB1 bobbin. The bobbin is started by bifilar winding the 175 turns. Bifilar winding is accomplished by taking two separate pieces of wire and winding them simultaneously. The finish of one winding is then soldered to the start of the other and often, as is the case in this implementation, the point of connection of the two wires (center tap) is brought out to a pin of the transformer. The remaining ends (start and finish) now comprise the winding. The transformer is now followed by $1\frac{1}{4}$ layers of insulating tape. The insulating tape used was the Permacel P-256 which forms a dielectric capable of withstanding 5 KV, this serves to protect the line card and the subscribers

from lightning induced surges. The 70 turns are then bifilar wound; this results in a well balanced transformer. The start of one winding should be connected to the finish of the other and brought out to a pin, thus creating a center tap on the line winding. The transformer is then finished with 1½ layer of insulating tape. The transformer thus designed gave satisfactory results in the lab and is characterized by the following:

Secondary inductance	Ls = 26 mH
Secondary leakage inductance	ls = 20 μH
Secondary winding resistance	Rs = 1.5Ω
Primary winding resistance	Rp = 2.7Ω

The capacitance between the two bifilar windings was measured to be 100 pF and this may be too high for certain applications. For this case the bifilar winding can be replaced by the cross winding technique shown in Figure 6a. The two windings are now wound in opposite directions, one wire is on top on the top side while the other is on top on the bottom side. This technique reduced the above mentioned capacitance to less than 50 pF.

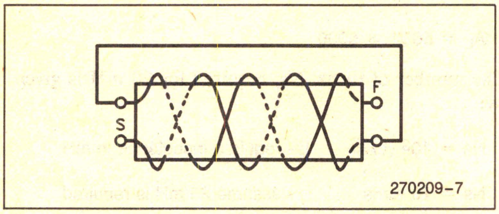


Figure 6a. Crosswinding

The 29C53 has been designed to drive voltages as specified in the I.430, since the transformer presents a series resistance, some of this voltage will be dropped across it. For the transformer designed above, the overall series resistance is (2.7 + 1.5*6.25) = 12Ω which will result in a 3.8% error over the allowed peak transmit signal in I.430. This is acceptable as I.430 allows a 10% error for the peak voltage. If series resistors are required to protect the Schottky diodes, their value may be calculated by having the maximum allowed peak voltage error. Note that equal value resistors should be placed on both arms of the line. If larger values of protection resistors are required, the above procedure may be repeated with a larger core. This will allow the same inductance to be achieved with a fewer turns and the larger core will make it possible to use a thicker wire. Both of these factors will contribute to reduce the winding resistance, hence a larger value diode protection resistor may be used. Alternatively, the transformer turns ratio can be decreased so that the output voltage is increased and hence more of it can be dropped across the series resistance. This in turn means that the value of this protection resistor can be increased. However, note that the 29C53 is only capable of driving loads greater than 200Ω. If a turns ratio of 2:1 is used then the overall series resistance can be 51Ω, or 31Ω in each arm. As discussed earlier this resistor can be larger on the 29C53 receive pins.

Some establishments may require further line isolation from the transformer in which case a Faraday shield can be placed in between the primary and the secondary windings. The Faraday shield can be made by wrapping 1¼ layers of a copper tape (such as the permacel P-389) between the two windings. The copper tape should be insulated from the windings and should be brought out to the local ground. As well as isolating, the Faraday shield also serves to reduce the interwinding capacitance.

The transformer designed was connected up as shown in Figure 6b to measure its longitudinal balance.

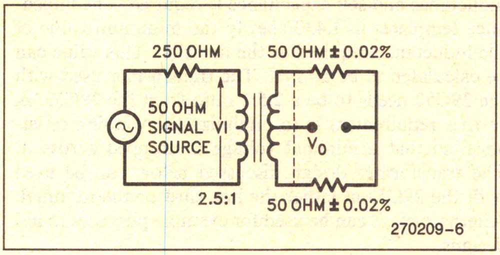


Figure 6b. Longitudinal Balance

Let v = vi/2.5
Then longitudinal balance is given by: 20 Log V/Vo

Measurements conducted showed this figure to be better than 70 dB for the frequency range of 10 KHz to 1 MHz.

The center tap on the primary (29C53 side) is coupled to ground via a 10 nF capacitor. In this manner longitudinal signals on the primary are bypassed to ground. Measurements produced greater than 70 dB of longitudinal signal rejection.

When designing the System board, special care should be paid to the layout. The transformer and the 29C53 should both be placed on a ground plane. The connecting tracks from the 29C53 to the transformer should be as short as possible. The two devices should be placed close to the edge where the transmission lines interface, while the high frequency logic should be placed on the opposite edge. The analog ground wiring should follow a star configuration and should have a separate isolated lead originating from the system ground where it enters the board.

Though the analysis of pulse transformers is beyond the scope of this brief (2), one should be aware of the pertinent parameters affecting the good reproduction of the pulse. The pulse transformer is generally analyzed by different equivalent circuits, depicting the varying phases of the pulse.

Figure 7 shows these circuits. The pulse shape is then optimized by considering the transient response of the equivalent circuits.

The pulse response of the transformer is characterized by a finite rise time, a decaying top period and finite fall time as depicted in Figure 7d. The fastest rise time that can be obtained without overshoot is for the critically damped case and is given by:

$$tr = 3.35\sqrt{\alpha Lc} \qquad \text{where } \alpha = R_L / (Rg + R_L)$$

For the top period, there will be some decay leading to a fractional droop, this is given by:

$$D \cong \tau \frac{Lp}{R} \qquad \text{where } \tau = \text{pulse width}$$

$$R = R_L \text{ and } Rg \text{ in parallel}$$

The fall period is characterized by the second order circuit of Figure 7c; the primary concern here to prevent severe undershoot or backswing when the 29C53 transmitter is in the high impedance mode. This can best be achieved by having an overdamped system, which is the case when:

$$Lp > 4CR_L^2$$

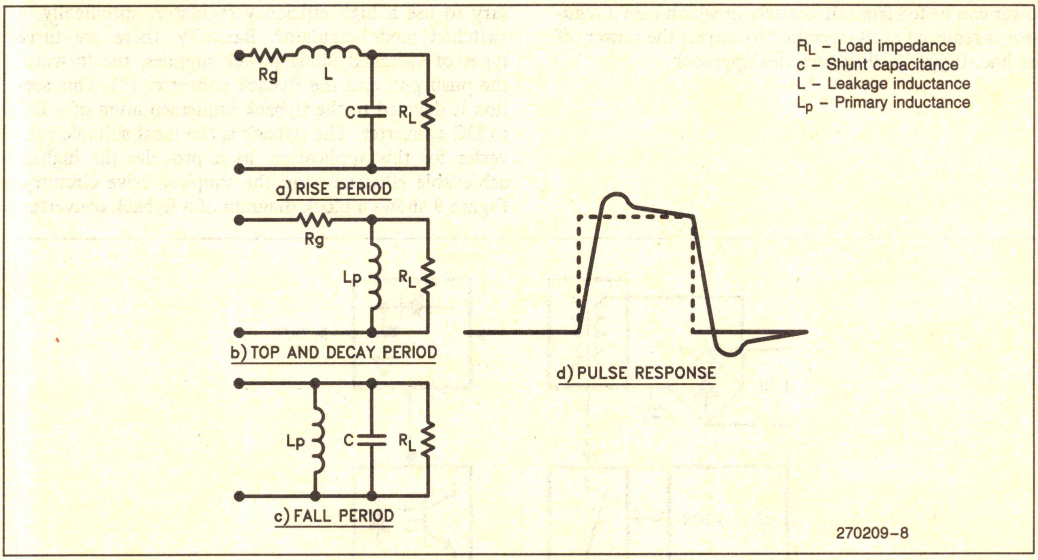


Figure 7. (a) Equivalent Circuits for Rise Period (b) Top and Decay Period (c) Fall Period (d) The Pulse Response

Commercially available pulse transformers exist which are compatible with the 29C53. Two examples are the AIE Magnetics 325-0172 and the CTM 25585. Most manufacturers will modify their design to meet the requirements of a particular application.

POWER EXTRACTION

The same transformer can be used at both the line card and the terminal, and the same protection scheme can be used at both ends of the loop. The need now arises to provide power to the terminal. There are a number of ways of providing power to the terminal, for instance a secondary cell can be used as battery back-up in conjunction with a main supply. There is also some scope for trickle charging secondary cells from the line or from a small solar cell array, but the drawback with secondary cells tends to be their short life span. This disadvantage can be offset by using special purpose primary cells as a back-up supply, these do not need any charging circuitry and can be expected to have life expectancy twice that of the secondary cells. Finally, the power can be fed from the switch, in which case a regulator is required at the terminal to extract the power off the line. Figure 8 illustrates this approach.

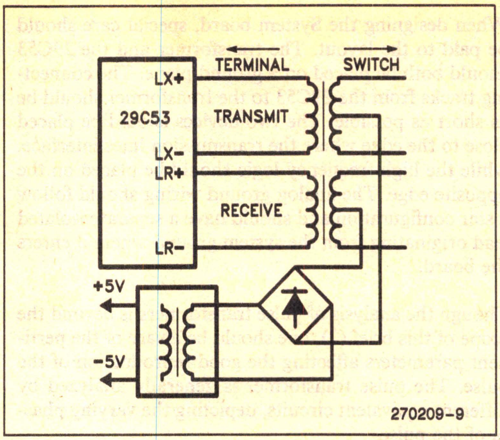


Figure 8. Power Extraction

A DC to DC converter is required to convert the line voltage to 5V for the local circuitry. In order to obtain the lowest losses in the conversion process, it is necessary to use a high efficiency regulator, specifically, a switched mode regulator. Basically, there are three types of switched mode power supplies, the forward, the push pull and the flyback converter (3). This section is devoted to the flyback implementation of a DC to DC converter. The flyback is the most suitable converter for this application, as it provides the highest achievable efficiency and the simplest drive circuitry. Figure 9 shows a block diagram of a flyback converter.

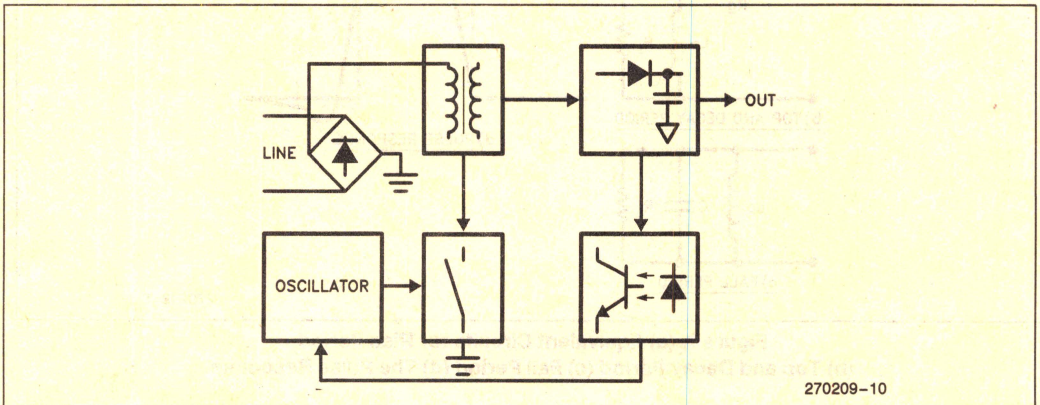


Figure 9. Flyback Converter

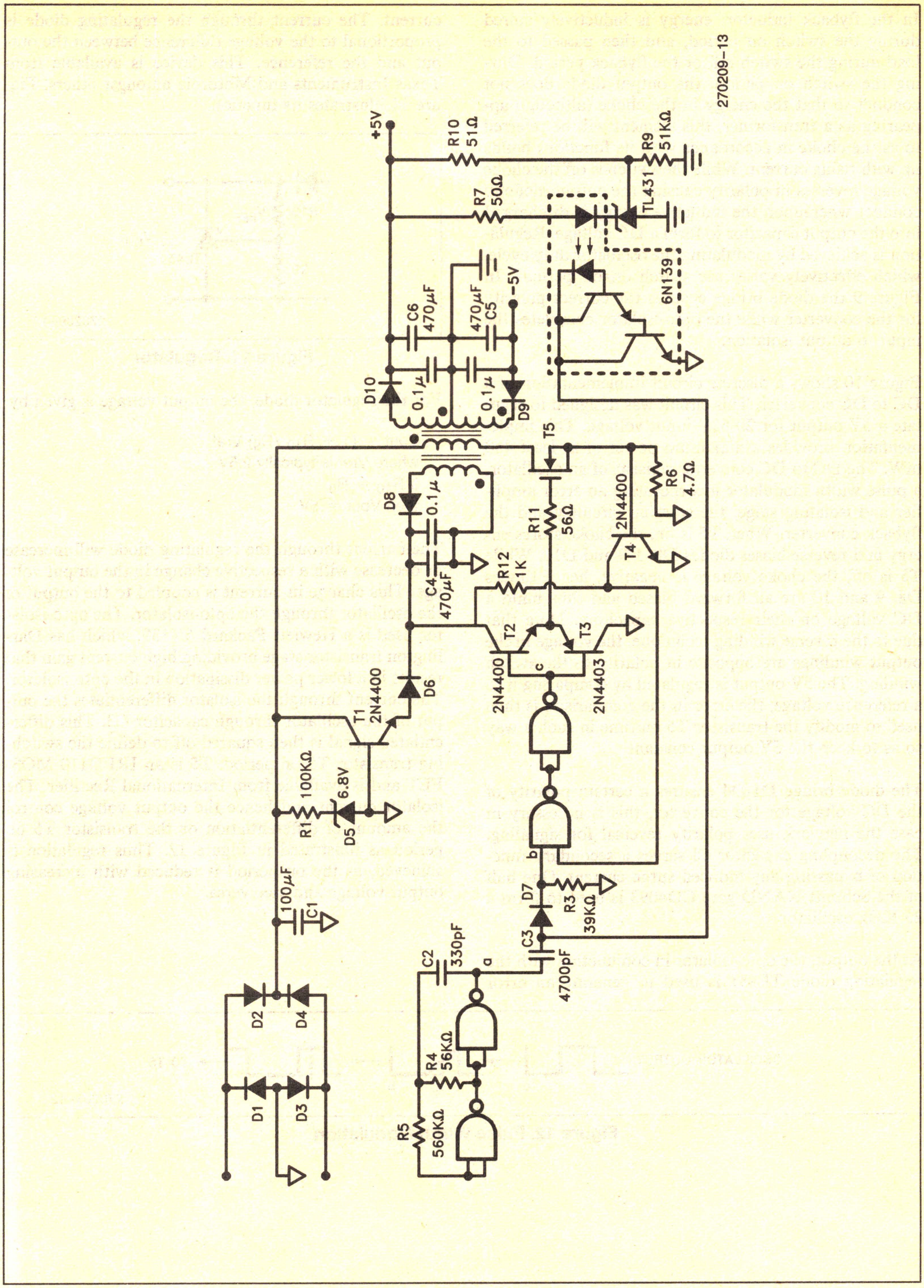


Figure 10. DC to DC Converter

In the flyback inductor, energy is inductively stored during the switch on period, and then passed to the load during the switch off, or the flyback period. During the switch on period, the output diode does not conduct so that the energy in the choke (although appearing as a transformer, this element will be referred to as the choke in accordance with its function) builds up with rising current. While the switch is off the choke voltage reverses in polarity causing the output diode to conduct whereupon the inductive energy is discharged into the output capacitor to form a DC voltage. Regulation is achieved by modulating the oscillator duty cycle, which effectively varies the switch on/off periods. In Figure 9 the diode bridge ensures the correct polarity for the converter while the opto-isolator completes the input to output isolation.

Figure 10 shows a discrete circuit implementation of a DC to DC converter. This circuit was designed to regulate a 5V output for 20-60V input voltage. This implementation provides a maximum power of at least 450 mW. The DC to DC converter consists of an oscillator, a pulse width modulator incorporating an error amplifier and isolating stage, the start up circuitry and the flyback converter. When T5 is on, the choke stores energy and reverse biases diodes D8, D9 and D10. While T5 is off, the choke voltage is negative, hence diodes D8, 9 and 10 are all forward biased and thus build a DC voltage on their respective capacitors. Note that due to the reverse winding technique, the voltage in the output windings are opposite in polarity to the switch winding. The 5V output is regulated by comparing it to a reference voltage, the error in the comparison is then used to modify the transistor T5 on time in such a way so as to keep the 5V output constant.

The diode bridge D1-D4 ensures a certain polarity of the DC voltage for the converter, this is necessary in case the network uses polarity reversal for signaling. The decoupling capacitor Cl serves a secondary function of bypassing any induced surge current. One half of the Schmitt NAND gate CD4093 is used to form a 25 KHz oscillator.

At the output, the opto-isolator in conjunction with the regulating diode TL431 is used to generate an error current. The current through the regulating diode is proportional to the voltage difference between the output and the reference. This device is available from Texas Instruments and Motorola amongst others. Figure 11 illustrates its function.

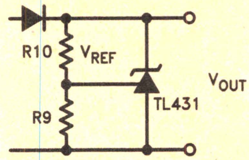


Figure 11. Regulator

For the regulator diode, the output voltage is given by:

$$Vout = (1 + R_{10}/R_9)\ Vref$$

where Vref is typically 2.5V.

If $R_{10} = R_9$

then Vout = 5V

The current through the regulating diode will increase or decrease with a respective change in the output voltage. This change in current is coupled to the output of the oscillator through the opto-isolator. The opto-isolator used is a Hewlett Packard 6N139, which has Darlington transistor stage providing high current gain that results in a lower power dissipation in the opto-isolator. The current through the isolator differentiates the output of the oscillator through capacitor C3. This differentiated signal is then squared off to define the switching transistor T5 on period. T5 is an IRFD110 MOSFET and is available from International Rectifier. The isolator current and hence the output voltage control the amount of differentiation or the transistor T5 on period as illustrated in Figure 12. Thus regulation is achieved, as the on period is reduced with increasing output voltage and vice versa.

Figure 12. Pulse Width Modulation

The two transistors T2 and T3 provide a low source impedance driving stage for the switching transistor. The fast current sinking and sourcing will ensure fast switching of transistor T5.

The input capacitance of the MOSFET IRFD110 is a maximum of 200 pF. Without the buffer stage the MOSFET will stay in the linear region longer before saturating, thus resulting in a slower switching speed. The slow switching in turn will result in a lower overall efficiency for the converter.

The resistor R6 and transistor T4 provide current overload protection. Transistor T4 will conduct when the voltage across R6 exceed 0.6V or conversely, the current through it is greater than 150 mA. With T4 conducting, the drive to the MOSFET is nulled by the associated NAND gate.

The transformer choke is a three winding transformer consisting of the switching winding, the output winding, which is split for the +5V and −5V and the self-bias winding. The transformer is designed for complete energy transfer under no load conditions and incomplete energy transfer under full load conditions. Figure 13 shows the wave forms of the two modes.

At full load, the incomplete energy transfer mode exhibits a lower peak switching transistor current, while the complete mode at lower power assures a smaller core. The inductance required to achieve this is 6.5 mH for the switch winding. The core used was an RM6CA400-3B7. The number of turns required to achieve this inductance is 130 and for a 20-60V line voltage, 50 turns are required for a +5V output, hence use 50 turns for the −5V too. The self bias winding uses 70 turns. The transformer was wound with 130 turns of 34AWG, followed by 50 bifilar turns of 32AWG and finished off with 70 turns of 32AWG. The dot scheme in Figure 10 should be adhered to. The bobbin is then immersed in varnish such as the Dolph's BC356 to dispel any moisture and to provide a protective coating. Alternatively, a commercially available DC to DC converter transformer such as the 326-0533 can be purchased from AIE Magnetics.

At start up, the converter is powered by the linear regulator D5, Rl and Tl, which sets the power supply at 5.3V. After start up the self bias winding forces the voltage on C4 to be between 7 and 15 volts, which will back bias diode D6, thus turning off the linear regulator. Under this condition the power supply provides a self bias voltage to keep it running, while little power is

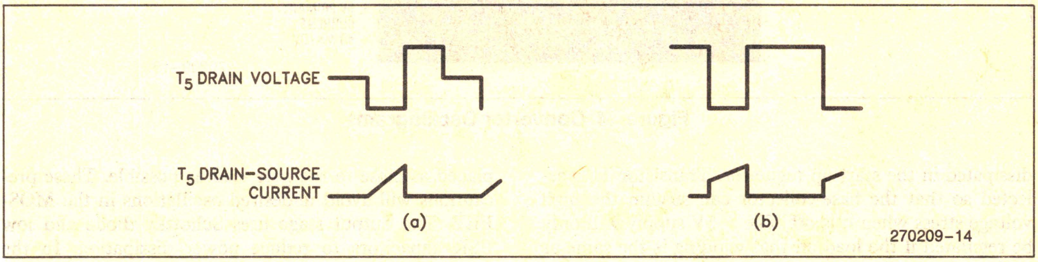


Figure 13. (a) Current Voltage Waveforms for Complete Energy Transfer (b) Waveforms for Incomplete Energy Transfer

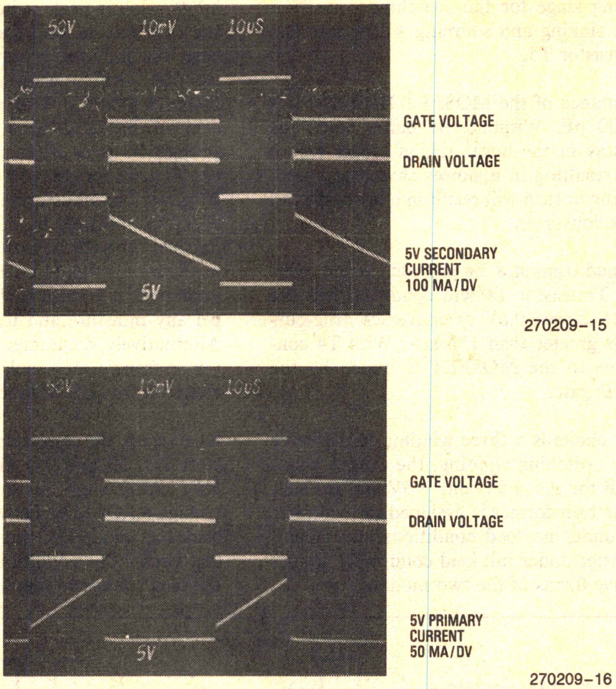


Figure 14. Converter Oscillograms

dissipated in the start up regulator. Transistor Tl is selected so that the base-collector can sustain the high voltage stress when it is off. The −5V supply will only be regulated if the load on that winding is the same as that on the +5V winding. If this is not possible, it may be necessary to use a linear post regulator to obtain a regulated −5V supply.

Figure 14 shows the volt-current oscillograms for a 30V line voltage and 400 mW output power. This shows the flyback converter working in the incomplete energy transfer mode. The results obtained in the lab gave an overall efficiency of better than 67% and a power supply ripple of less than 25 mV. The no load power consumption was less than 50 mW. Regulation of the output voltage was better than 150 mV.

The design was wire wrapped to illustrate the concept of power extraction and can of course, be optimized for better performance. Special care should be paid to the layout; Figure 15 shows good layout principles. Use star ground connections to avoid current loops in the ground.

All lead lengths going to the switching MOSFET should be minimized and in particular the gate lead. The resistor in series with the MOSFET should be placed as close to the gate lead as possible. These precautions will avoid undesired oscillations in the MOSFET. The output stage uses Schottky diode and low ESR capacitors to reduce power dissipation. In the event of any undesired EMI radiation the transformer can be placed in an electromagnetic container and the converter can be enclosed in a copper container.

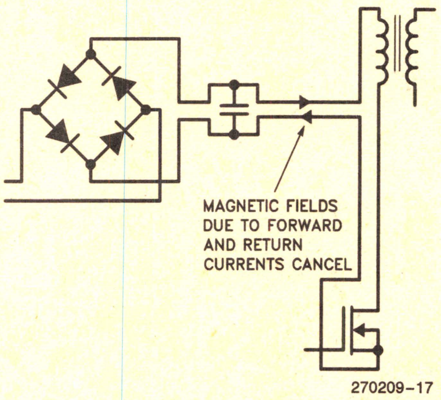


Figure 15. Good Layout Principles

POWER FAILURE CONSIDERATIONS

Without power the line interface pins of the 29C53 appear as diode drops across the line. This means that the transmitter of the Network Terminator and the powered on terminals in a multidrop configuration will be terminated by a diode instead of the usual 50Ω. In the event of a failure, it therefore becomes necessary to isolate the offending terminal from the line. This can be done by providing a switch in the transmit path that is normally closed and opens when no power is applied.

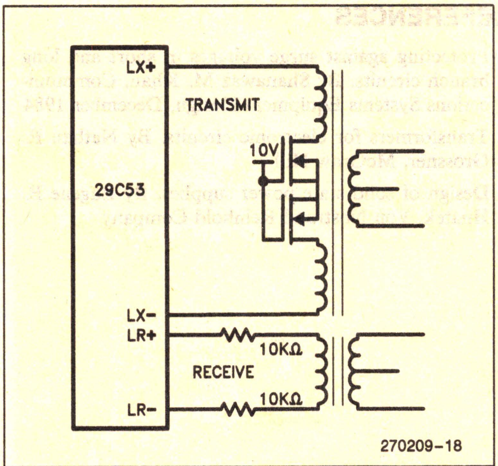


Figure 16. Isolation of Equipment in Case of Power Failure

In the receive path, it is only necessary to increase the impedance seen by the line. One way to implement this principle is to use a MOSFET bilateral switch in the transmit path and to place series resistors in the receive path, such that the impedance seen by the line is greater than 2500Ω. Figure 16 illustrates this approach. A noteworthy point is that the series resistors in the receive path not only provide terminal isolation in case of failure but also protect the terminal from current surges.

When there is power, the two MOSFETS will be on and appear as a small on resistance, which has to be included in the line transformer design analysis. When there is no power, the MOSFETS appear as back to back diodes, thus stopping any AC flow. The VN0300 MOSFETS manufactured by Siliconix may be used, when on they present a 1.2Ω resistance each. Note that in order to ensure that the MOSFETS conduct it is necessary to have a 10V supply in the system. If this is not possible the MOSFETS can be replaced by a Reed relay which presents a lower on resistance and capacitance but has the disadvantage of consuming more power. A low power relay could not be located hence a vendor was requested to customize one. Figure 17 shows the isolation technique using the Wabash 1992-2-1 25 mW relay which will operate at 3.8V and release at 0.5V.

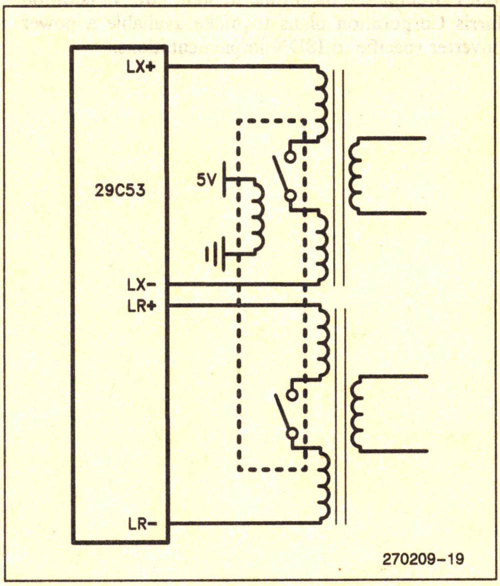


Figure 17. Power Failure Isolation

CONCLUSION

Specific implementations have been provided for the general aspects of line interfacing at both the line card and the terminal end. These solutions can be taken as they are and placed in the particular application or used to aid a system design.

The fixed voltage or constant current feed are both simpler and more economical to realize in discrete form; however the constant current variable voltage scheme may be more suitable in an integrated form. The power converter discussed was based on a low cost simple implementation and it is certainly possible to optimize it to obtain conversion efficiencies in the 75% range. As an alternative to discrete implementations, a low power switch mode power supply is commercially available from Fairchild and Motorola, to name two. In addition Harris Corporation plans to make available a power converter specific to ISDN implementations.

The protection circuits and the transformer, however, can only be provided in discrete form at the present time. The concepts presented in the protection section emphasized low capacitance and maximum protection. The section took an overkill approach and as such a subset of the discussed ideas should suffice most applications. The transformer designed pointed out the relevant parameters to consider and can be used as it is or modified to the particular application. Of course the ISDN transformer is also commercially available.

REFERENCES

1. Protecting against surge voltages in short and long branch circuits. By Shanawaz M. Khan, Communications Systems Equipment Design, December 1984
2. Transformers for electronic circuits. By Nathan R. Grossner, McGraw Hill
3. Design of solid state power supplies. By Eugene R. Hnatek, Von Nostrand Reinhold Company

December 1986

ISDN Applications with 29C53 and 80188

HERBERT WEBER
TELECOM OPERATIONS

Order Number: 270247-001

TERMINAL ADAPTOR (TA)

A terminal adaptor, or "TA", is the link between existing non-ISDN equipment like terminals, facsimile, printers and the ISDN network. The function of this application is to effectively replace equipment such as a modem. Usually provided as a separate box, it processes RS232 or X.27 data and places it on the 4 wire 'S' loop. No change at the terminal is required to make it ISDN compatible.

The design is based on a 29C53 transceiver for the ISDN connection and an 80188 microprocessor in combination with an 82530 communications controller for the data connection. Benefits of the application are:

- Data rates up to 19.2 Kb/s using an RS232 interface or up to 48 Kb/s using an X.21 interface.
- Compact design and low cost.
- Virtually error free transmission.

Link Setup

The user sets up a call in the same manner as a Hayes* modem user does, i.e. a command is transferred to the adaptor via the RS232 interface. The command takes the form of an ASCII string in which the first 2 characters are "AT".

The 80188 accepts the command and begins the call setup procedure by communicating the call's destination to the NT (or CO). This is achieved by passing call setup messages to a link level protocol, which is passed to the NT over the physical level (S bus). The partitioning of the tasks is as follows:

82530

Full duplex, dual channel serial communications controller capable of working in asynchronous, bit or byte synchronous modes. The 82530 receives commands from the terminal's RS232 or X.21 interface and passes them on to the 80188.

80188

After having received the dialing information from the keyboard, the 80188 sets up the call via the 29C53 D-channel by sending the appropriate CCITT message up the link.

- Call setup message generation (CCITT I.451).

*Hayes is a registered trademark of Hayes Microcomputer Products, Inc.

- Upper portion of link access procedure (CCITT I.440) handling:
  - Multiple logic channels
  - Sequence control
  - Error correction (retransmission)
  - Flow control

EPLD

- Interface conversion, serial to/from SLD
- B-channel assignment

29C53

- Physical level interface (CCITT I.430)
- Lower portion of the link access procedure:
  - Zero insertion/deletion
  - CRC generation and checking
  - Flag appending and detection
- D-channel message buffering

The 80188 passes the information for the D-channel messages via the parallel bus into the FIFO's of the 29C53.

The NT grants a B-channel (if available) to the TA and the channel is now ready for data transfer.

Data Transfer

An indication is given to the user's terminal via the RS232 or X.21 port that communication may commence. Any subsequent data, from the terminal, is treated as follows:

Data from the terminal passes via the 82530 to RAM via one of the 80188 DMA channels. D-channel communication is supported by its own dedicated DMA channel.

The 80188 fetches the data from RAM, depacketizes and packetizes it before sending it back to the 82530 where a protective HDLC protocol is added.

From the 82530 the data reaches the EPLD to be inserted into the B1 or B2 channel on the SLD bus. The 29C53 sends it out over the "S" interface.

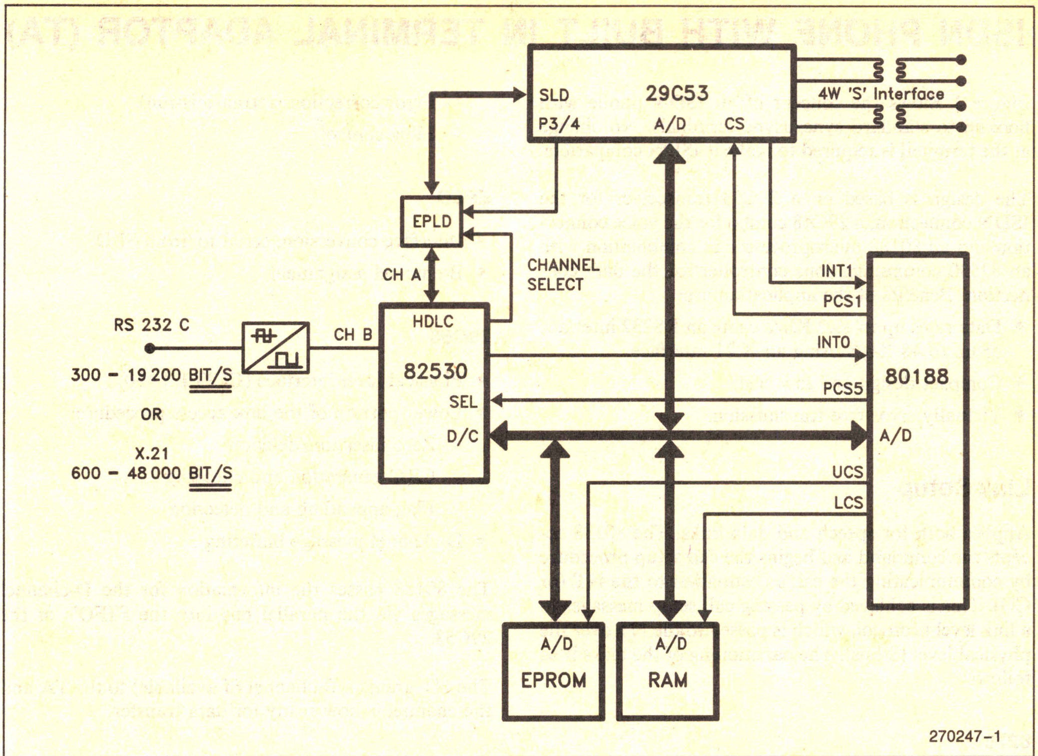


Figure 1. Terminal Adaptor

ISDN PHONE WITH BUILT IN TERMINAL ADAPTOR (TA)

Figure 1 shows the concept of an ISDN phone with hookup to standard sync/async terminals. No change at the terminal is required to make it ISDN compatible.

The design is based on a 29C53 transceiver for the ISDN connection, a 29C48 combo for the voice connection and an 80188 microprocessor in combination with an 82530 communications controller for the data connection. Benefits of the application are:

- Data rates up to 19.2 Kb/s using an RS232 interface or up to 48 Kb/s using an X.21 interface.
- Compact design and low cost.
- Virtually error free transmission.

Link Setup

Applies both for speech and data links. The 80188 accepts the command and begins the call setup procedure by communicating the call's destination to the NT (or CO). This is achieved by passing call setup messages to a link level protocol, which is passed to the NT over the physical level (S-bus). The partitioning of the tasks is as follows:

8279

The 8279 keyboard and display controller scans the telephone number pad and supports a small telephone display. Calls are initiated either through the terminal keyboard using an extended Hayes Smart Modem command set or via the telephone number pad.

82530

Full duplex, dual channel serial communications controller capable of working in asynchronous, bit or byte synchronous modes. The 82530 receives commands from the terminal's RS232 or X.21 interface and passes them on to the 80188.

80188

After having received the dialing information from either keyboard, the 80188 sets up the call via the 29C53 D-channel by sending the appropriate message up the link.

- Call setup message generation (CCITT I.451)
- Upper portion of link access procedure (CCITT I.440) handling:
  - — Multiple logic channels
  - — Sequence control
  - — Error correction (retransmission)
  - — Flow control

EPLD

- Interface conversion, serial to/from SLD
- B-channel assignment

29C53

- Physical level interface (CCITT I.430)
- Lower portion of the link access procedure:
  - — Zero insertion/deletion
  - — CRC generation and checking
  - — Flag appending and detection
- D-channel message buffering

The 80188 passes the information for the D-channel messages via the parallel bus into the FIFO's of the 29C53.

The NT grants a B-channel (if available) to the TA and the channel is now ready for data transfer.

Information Transfer

VOICE

The voice transfer is supported by the 29C48 which transmits the voice on either the B1 or B2 channel (controlled by the EPLD) into the 29C53 and onward to the S-bus.

DATA

An indication is given to the user's terminal via the RS232 or X.21 port that communication may commence. Any subsequent data, from the terminal, is treated as follows:

Data from the terminal passes via the 82530 to RAM via one of the 80188 DMA channels. D-channel communication is supported by its own dedicated DMA channel.

The 80188 fetches the data from RAM, depacketizes and packetizes it before sending it back to the 82530 where a protective HDLC protocol is added.

From the 82530, the data reaches the EPLD to be inserted into the B1 or B2 channel on the SLD bus. The 29C53 sends it out over the "S" interface.

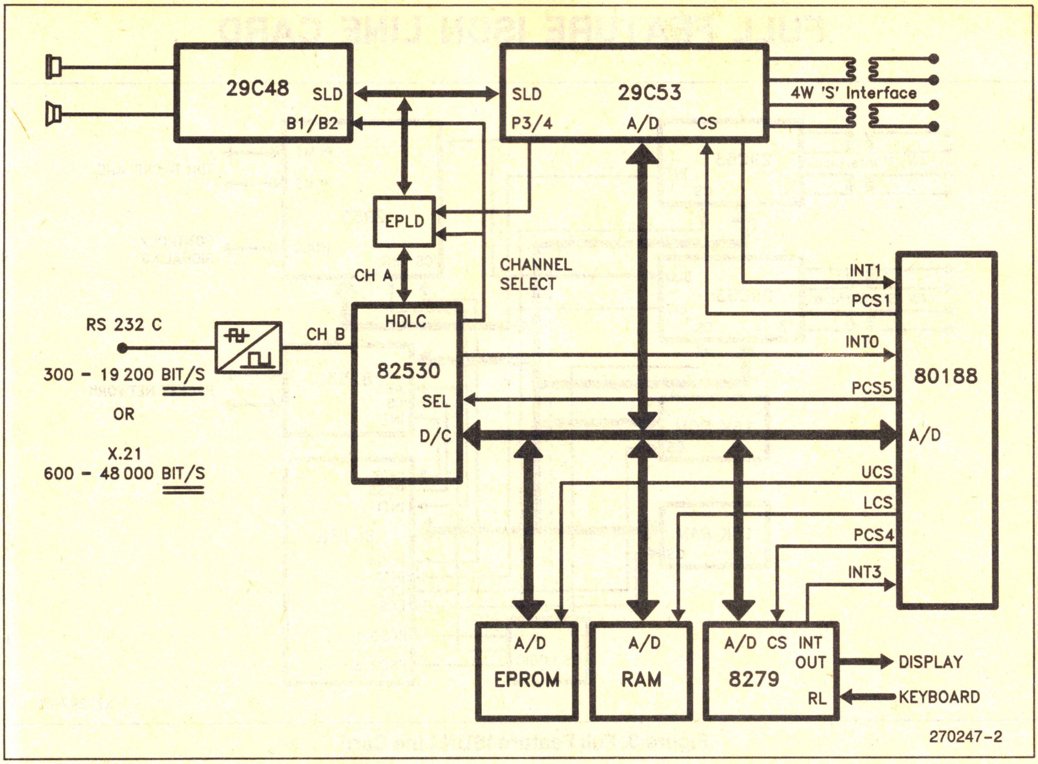


Figure 2. ISDN Phone With Built In Terminal Adaptor

FULL FEATURE ISDN LINE CARD

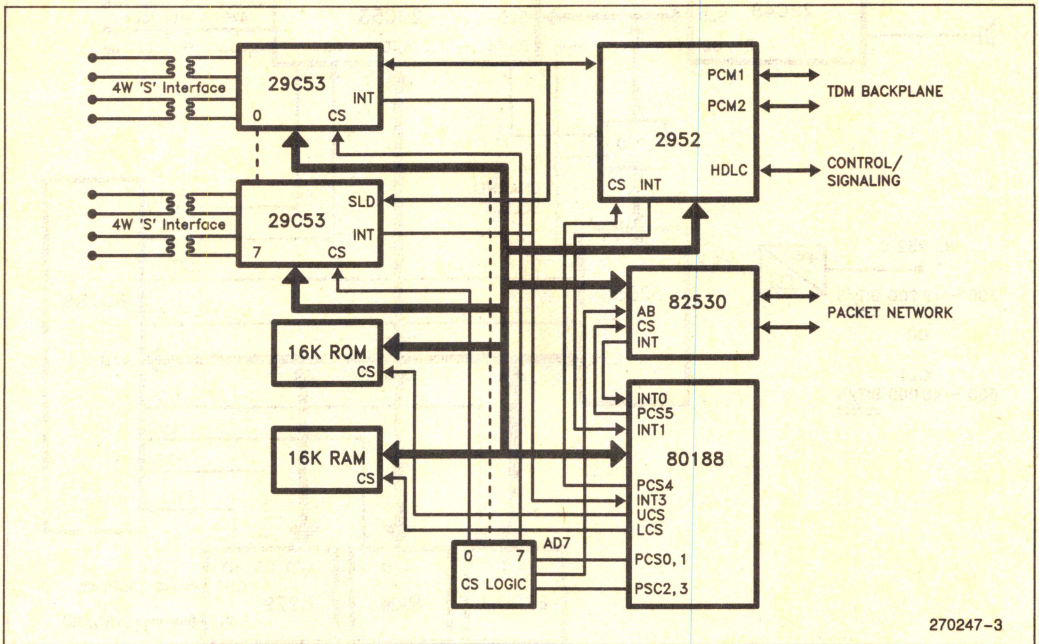


Figure 3. Full Feature ISDN Line Card

The addition of ISDN line cards to a PABX provides the user with access to the ISDN network. While the analog line card provides access for standard telephone as well as for modems, ISDN terminal adaptors, terminals and phones are connected to the ISDN line card via a 4 wire 'S' loop. The described application provides all functions to separate voice and switched data (B-channels) from signaling and packetized data (D-channel).

Intel's 2952 Line Card Controller provides the foundation for the PBX backplane interface, allowing eight 29C53's and with it, eight "S" interface loops, to hook up in parallel to the line card.

The 29C53 and 80188 together handle the processing of D-channel protocols and messages as follows:

1. The 29C53 executes all bit level HDLC processing, puts the "raw" messages into its FIFO and raises the interrupt signal.
2. A special status register in the 29C53s allows the 80188, through a single status read operation, to determine which of the 29C53s is requiring interrupt servicing, i.e. has D-channel messages(s) in its FIFO.
3. The 80188's DMA controller accesses the FIFO concerned and the data is transferred to RAM without CPU involvement.
4. The 80188 determines whether the data is for signaling (S-packet) or is a message to be sent out over the packet (P-packet) switched network.

   Signaling information can be processed locally or sent via the 2952 either interleaved on the PCM highway or separately over the HDLC serial control channel.

   If the data is of "P" type, meant for the packet switched network, it is DMA'd into the 82530 serial communications controller which performs the necessary HDLC transmission, again without any CPU involvement.
5. The 80188 software is responsible for sending out acknowledgements for received messages from the 29C53's D-channel and can thus support large window sizes.

B-channel information is directly passed from the "S" loop over the 29C53 and 2952 to the switch backplane.

For transmission in the opposite direction, the procedure is equivalent to the one described above.

OTHER AVAILABLE TELECOM LITERATURE

Title	Order Number
2952 Reference Manual	270065-001
iATC 2952 Line Card Controller Reference Card	270059-001
Line Card Development Kit Users Guide	
Analog Line Card Designs Using iATC Components	
29C53 Reference Manual	270151-001
29C53 Line Card Evaluation Kit (LEK) Manual	
29C53 Terminal Evaluation Kit (TEK) Manual	

PCM Codec/Filter and Combo

6

2910A
PCM CODEC - μLAW
8-BIT COMPANDED A/D AND D/A CONVERTER

- **Per Channel, Single Chip Codec**
- **CCITT G711 and G712 Compatible, ATT T1 Compatible with 8th Bit Signaling**
- **Microcomputer Interface with On-Chip Timeslot Computation**
- **Simple Direct Mode Interface When Fixed Timeslots are Used**
- **±5% Power Supplies: +12V, +5V, −5V**
- **78dB Dynamic Range, with Resolution Equivalent to 12-Bit Linear Conversion Around Zero**
- **Precision On-Chip Voltage Reference**
- **Low Power Consumption 230 mW Typ. Standby Power 33mW Typ.**
- **Fabricated with Reliable N-Channel MOS Process**

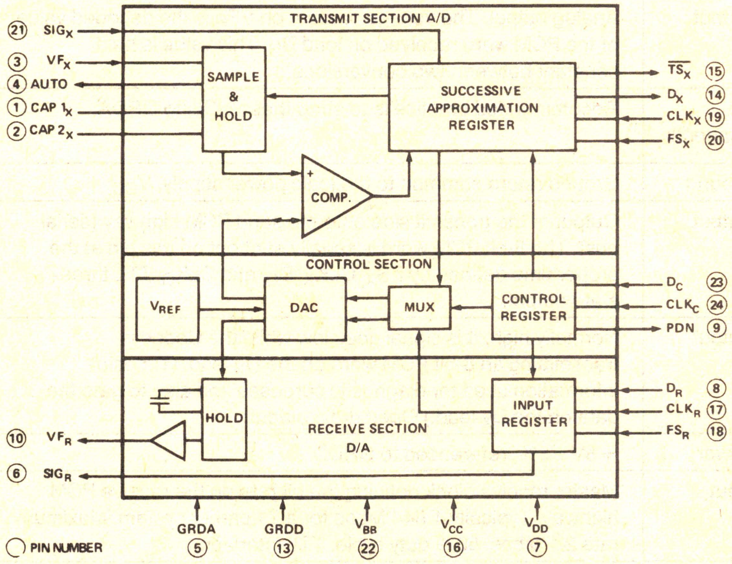
The Intel 2910A is a fully integrated PCM (Pulse Code Modulation) Codec (Coder-Decoder), fabricated with N-channel silicon gate technology. The high density of integration allows the sample and hold circuits, the digital-to-analog converter, the comparator and the successive approximation register to be integrated on the same chip, along with the logic necessary to interface a full duplex PCM link and provide in-band signaling.

The primary applications are in telephone systems:

- Transmission —T1 Carrier
- Switching —Digital PBX's and Central Office Switching Systems
- Concentration —Subscriber Carrier/Concentrators

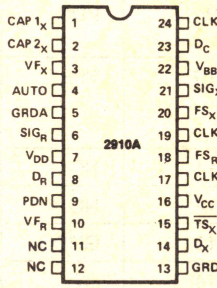
The wide dynamic range of the 2910A (78dB) and the minimal conversion time (80μsec minimum) make it an ideal product for other applications, like:

- Date Acquisition • Telemetry • Secure Communications Systems • Signal Processing Systems



006785-2

Figure 1. Block Diagram



006785-1

Figure 2. Pin Configuration

CAP 1_X, CAP 2_X	Holding Capacitor
VF_X	Analog Input
VF_R	Analog Output
D_R, D_C, SIG_X	Digital Input
SIG_R, D_X, $\overline{TS_X}$	Digital Output
CLK_C, CLK_X, CLK_R	Clock Input
FS_X, FS_R	Frame Sync Input
AUTO	Auto Zero Output
V_{BB}	Power (−5V)
V_{CC}	Power (+5V)
V_{DD}	Power (+12V)
PDN	Power Down
GRDA	Analog Ground
GRDD	Digital Ground
NC	No Connect

Figure 3. Pin Names

November 1986
Order Number: 006785-002

Pin Description

Pin No.	Symbol	Function	Description
1	$CAP1_X$	Hold	Connections for the transmit holding capacitor. Refer to Applications section.
2	$CAP2_X$		
3	VF_X	Input	Analog input to be encoded into a PCM word. The signal on this lead is sampled at the same rate as the transmit frame synchronization pulse FS_X, and the sample value is held in the external capacitor connected to the $CAP1_X$ and $CAP2_X$ leads until the encoding process is completed.
4	AUTO	Output	Most significant bit of the encoded PCM word (+5V for negative, −5V for positive inputs). Refer to the Codec Applications section.
5	GRDA	Ground	Analog return common to the transmit and receive analog circuits. Not connected to GRDD internally.
6	SIG_R	Output	Signaling output. SIG_R is updated with the 8th bit of the receive PCM word on signaling frames, and is latched between two signaling frames. TTL interface.
7	V_{DD}	Power	$+12V \pm 5\%$; referenced to GRDA.
8	D_R	Input	Receive PCM highway (serial bus) interface. The Codec serially receives a PCM word (8 bits) through this lead at the proper time defined by FS_R, CLK_R, D_C, and CLK_C.
9	PDN	Output	Active high when Codec is in the power down state. Open drain output.
10	VF_R	Output	Analog output. The voltage present on VF_R is the decoded value of the PCM word received on lead D_R. This value is held constant between two conversions.
11	NC	No Connects	Recommended practice is to strap these NC's to GRDA.
12	NC		
13	GRDD	Ground	Ground return common to the logic power supply, V_{CC}.
14	D_X	Output	Output of the transmit side onto the send PCM highway (serial bus). The 8-bit PCM word is serially sent out on this pin at the proper time defined by FS_X, CLK_X, D_C, and CLK_C. TTL three-state output.
15	$\overline{TS}_X$	Output	Normally high, this signal goes low while the Codec is transmitting an 8-bit PCM word on the D_X lead. (Timeslot information used for diagnostic purposes and also to gate the data on the D_X lead.) Open drain output.
16	V_{CC}	Power	$+5V \pm 5\%$, referenced to GRDD.
17	CLK_R	Input	Master receive clock defining the bit rate on the receive PCM highway. Typically 1.544 Mbps for a T1 carrier system. Maximum rate 2.1 Mbps. 50% duty cycle. TTL interface.
18	FS_R	Input	Frame synchronization pulse for the receive PCM highway. Resets the on-chip timeslot counter for the receive side. Maximum repetition rate 12 KHz. Also used to differentiate between non-signaling frames and signaling frames on the receive side. TTL interface.

Pin Description (Continued)

Pin No.	Symbol	Function	Description
19	CLK_X	Input	Master transmit clock defining the bit rate on the transmit PCM highway. Typically 1.544 Mbps for a T1 carrier system. Maximum rate 2.1 Mbps. 50% duty cycle. TTL interface.
20	FS_X	Input	Frame synchronization pulse for the transmit PCM highway. Resets the on-chip timeslot counter for the transmit side. Maximum repetition rate 12 KHz. Also used to differentiate between non-signaling frames and signaling frames on the transmit side. TTL interface.
21	SIG_X	Input	Signaling input. This digital input is transmitted as the 8th bit of the PCM word on the D_X lead, on signaling frames. TTL interface.
22	V_{BB}	Power	$-5V \pm 5\%$, referenced to GRDA.
23	D_C	Input	Data input to program the Codec for the chosen mode of operation. Becomes an active low chip select when CLK_C is tied to V_{CC}. TTL interface.
24	CLK_C	Input	Clock input to clock in the data on the D_C lead when the timeslot assignment feature is used; tied to V_{CC} to disable this feature. TTL interface.

FUNCTIONAL DESCRIPTION

The 2910A PCM Codec provides the analog-to-digital and the digital-to-analog conversions necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system.

In a typical telephone system the Codec is used between the PCM highways and the channel filters.

The Codec provides two major functions:

- Encoding and decoding of analog signals (voice and call progress tones)
- Encoding and decoding of the signaling and supervision information

On a non-signaling frame, the Codec encodes the incoming analog signal at the frame rate (FS_X) into an 8-bit PCM word which is sent out on the D_X lead at the proper time. Similarly, the Codec fetches an 8-bit PCM word from the receive highway (D_R lead) and decodes an analog value which will remain constant on lead VF_R until the next receive frame. Transmit and receive frames are independent. They can be asynchronous (transmission) or synchronous (switching) with each other.

For channel associated signaling, the Codec transmit side will encode the incoming analog signal as previously described and substitute the signal present on lead SIG_X for the least significant bit of the encoded PCM word. Similarly, on a receive signaling frame, the Codec will decode the 7 most significant bits according to the CCITT G733 recommendation and will output the least significant bit value on the SIG_R lead until the next signaling frame. Signaling frames on the send and receive sides are independent of each other, and are selected by a double-width frame sync pulse on the appropriate channel.

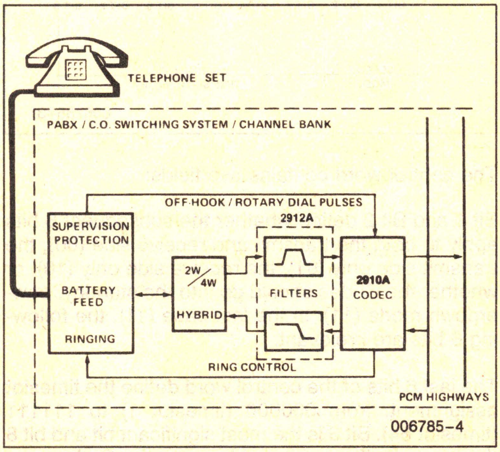


Figure 4. Typical Line Termination

The 2910A Codec is intended to be used on line and trunk terminations. The call progress tones (dial tone, busy tone, ring-back tone, re-order tone), and the prerecorded announcements, can be sent through the voice-path; digital signaling (off hook and disconnect supervision, rotary dial pulses, ring control) is sent through the signaling path.

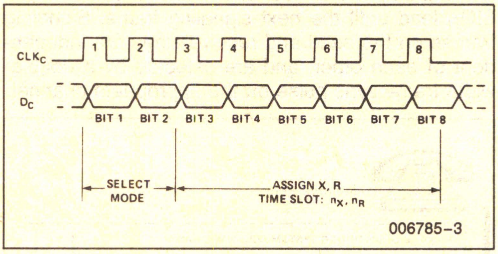
Circuitry is provided within the Codec to internally define the transmit and receive timeslots. In small systems this may eliminate the need for any external timeslot exchange; in large systems it provides one level of concentration. This feature can be bypassed and discrete timeslots sent to each Codec within a system.

In the power-down mode, most functions of the Codec are directly disabled to reduce power dissipation to a minimum.

CODEC OPERATION

Codec Control

The operation of the 2910A is defined by serially loading an 8-bit word through the D_C lead (data) and the CLK_C lead (clock). The loading is asynchronous with the other operations of the Codec, and takes place whenever transitions occur on the CLK_C lead. The D_C input is loaded in during the trailing edge of the CLK_C input.



The control word contains two fields:

Bit 1 and Bit 2 define whether the subsequent 6 bits apply to both the transmit and receive side (00), the transmit side only (01), the receive side only (10), or whether the Codec should go into the standby, powerdown mode (11). In the last case (11), the following 6 bits are irrelevant.

The last 6 bits of the control word define the timeslot assignment, from 000000 (timeslot 1) to 111111 (timeslot 64). Bit 3 is the most significant bit and bit 8 the least significant bit and last into the Codec.

Bit 1	Bit 2	Mode
0	0	X & R
0	1	X
1	0	R
1	1	Standby

Bit						
3	4	5	6	7	8	Timeslot
0	0	0	0	0	0	1
0	0	0	0	0	1	2
			•			•
			•			•
			•			•
			•			•
1	1	1	1	1	1	64

The Codec will retain the control word (or words) until a new word is loaded in or until power is lost. This feature permits dynamic allocation of timeslots for switching applications.

Microcomputer Control Mode

In the microcomputer mode, each Codec performs its own timeslot computation independently for the transmit and receive channels by counting clock pulses (CLK_X and CLK_R). All Codecs tied to the same data bus receive identical framing pulses (FS_X and FS_R). The framing pulses reset the on-chip timeslot counters every frame; hence the timeslot counters of all devices are synchronized. Each Codec is programmed via CLK_C and D_C for the desired transmit and receive timeslots according to the description in the Codec Control Section. All Codecs tied to the same D_R bus will, in general, have different receive timeslots, although that is not a device requirement. There may be separate busses for transmit and receive or all Codecs may transmit and receive over the same bus, in which case the transmit and receive channels must be synchronous ($CLK_X = CLK_R$). There are no other restrictions on timeslot assignments; a device may have the same transmit and receive timeslot even if a single bus is used.

There are several requirements for using the CLK_C-D_C interface in the microcomputer mode.

1) A complete timeslot assignment, consisting of eight negative transitions of CLK_C, must be made in less than one frame period. The assignment

can overlap a framing pulse so long as all 8 control bits are clocked in within a total span of 125 μs (for an 8 KHz frame rate). CLK_C must be left at a TTL low level when not assigning a timeslot.

2) A dead period of two frames must always be observed between successive timeslot assignments. The two frame delay is measured from the rising edge of the first CLK_C transition of the previous timeslot assigned.

3) When the device is in the power-down state (Standby), the following three-step sequence must be followed to power-up the codec to avoid contention on the transmit PCM highway.

   a) Assign a dummy transmit timeslot. The dummy should be at least two timeslots greater than the maximum valid system timeslot (usually 24 or 32). For example, in a 24 timeslot system, the dummy could be any timeslot between 26 and 64. This will power-up the transmit side, but prevent any spurious D_X or $\overline{TS}_X$ outputs.

   b) Two frames later, assign the desired transmit timeslot.

   c) Two frames later assign the desired receive timeslot.

4) Initialization sequence: The device contains an on-chip power-on clear function which guarantees that with proper sequencing of the supplies (V_{CC} or V_{DD} on last), the device will initialize with no timeslot assigned to either the transmit or receive channel. After a supply failure or whenever the supplies are applied, it is recommended that either power down assignment be made first, or the first timeslot assignment be a transmit timeslot or a transmit/receive timeslot. The consequence of making a receive timeslot assignment first, after supply application, is that the transmit channel will assume timeslot 1, potentially producing bus contention.

5) Transmit only/receive only operation is permitted provided that a power down assignment is made first. Otherwise, special circuits which use only one channel should be physically disconnected from the unused bus; this allows a timeslot to be made to an unused channel without consequence.

6) Both frame synchronizing pulses (FS_X, FS_R) must be active at all times after power on clear (after power supplies are turned on). This requirement must be met during powerdown and receive only or transmit only operation, as well as during normal transmit and receive operation.

Example of Microcomputer Control Mode:

The two words 01000001 and 10000010 have been loaded into the Codec. The transmit side is now programmed for timeslot 2 and the receive side for timeslot 3. The Codec will output a PCM word on the transmit PCM highway (bus) during the timeslot 2 of the transmit frame, and will fetch a PCM word from the receive PCM highway during timeslot 3.

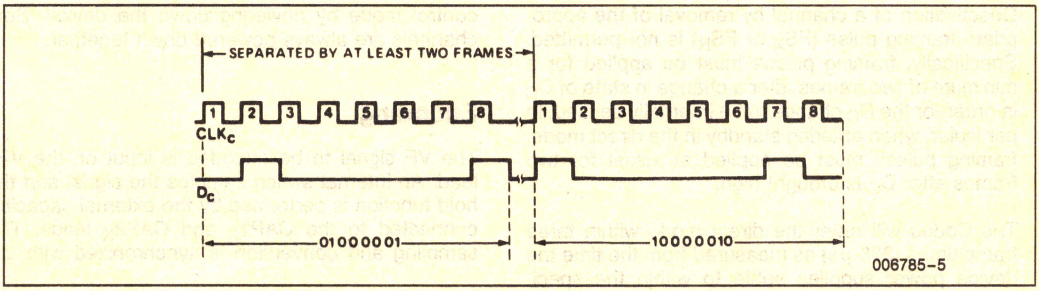


Figure 5. Microcomputer Mode Programming Example

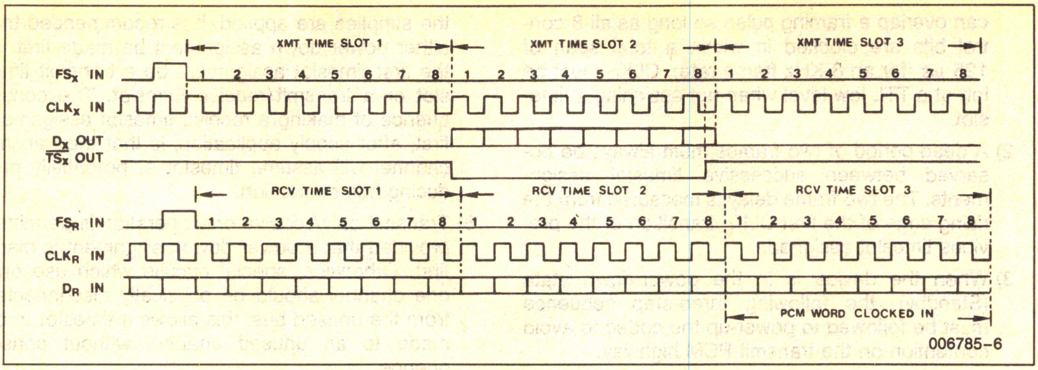


Figure 6. Microcomputer Mode PCM Highway Example

In this example the Codec interface to the PCM highway then functions as shown above. (FS_X and FS_R may be asynchronous.)

Direct Control Mode

The direct mode of operation will be selected when the CLK_C pin is strapped to the +5 volt supply (V_{CC}). In this mode, the D_C pin is an active low chip select. In other words, when D_C is low, the device transmits and receives in the timeslots which follow the appropriate framing pulses. With D_C high the device is in the power down state. Even though CLK_C characteristics are simpler for the 2910A it will operate properly when plugged into a 2910 board.

Deactivation of a channel by removal of the appropriate framing pulse (FS_X or FS_R) is not permitted. Specifically, framing pulses must be applied for a minimum of two frames after a change in state of D_C in order for the D_C change to be internally sensed. In particular, when entering standby in the direct mode, framing pulses must be applied as usual for two frames after D_C is brought high.

The Codec will enter the direct mode within three frame times (375 μs) as measured from the time the device power supplies settle to within the specified limits. This assumes that CLK_C is tied to V_{CC} and that all clocks are available at the time the supplies have settled.

General Control Requirements

All bit and frame clocks should be applied whenever the device is active. In particular, an unused channel cannot be deactivated by removal of its associated frame or bit clock while the other channel of the same device remains active.

A single channel cannot be deactivated except by physical disconnection of the data lead (D_X or D_R) from the system data bus. A device (both transmit and receive channels) may be deactivated in either control mode by powering down the device. Both channels are always powered down together.

Encoding

The VF signal to be encoded is input on the VF_X lead. An internal switch samples the signal and the hold function is performed by the external capacitor connected to the $CAP1_X$ and $CAP2_X$ leads. The sampling and conversion is synchronized with the

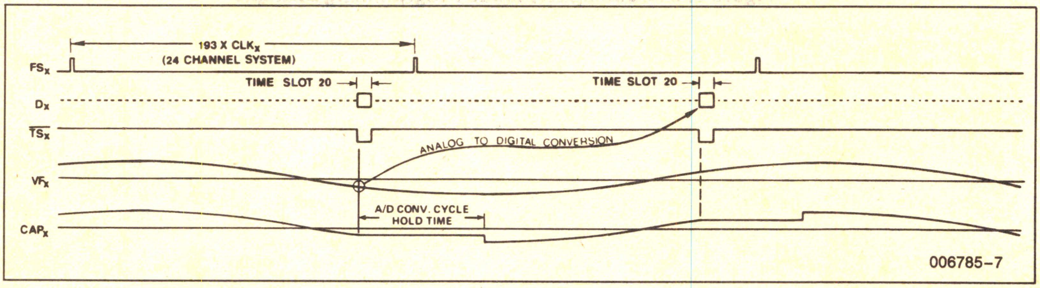


Figure 7. Transmit Encoding

transmit timeslot. The PCM word is then output on the D_X lead at the proper timeslot occurrence of the following frame. The A/D converter saturates at approximately ±2.2 volts RMS (±3.1 volts peak).

Decoding

The PCM word is fetched by the D_R lead from the PCM highway at the proper timeslot occurrence. The decoded value is held on an internal sample and hold capacitor. The buffered non-return to zero output signal on the VF_R lead has a dynamic range of approximately ±2.2 volts RMS (±3.1 volts peak).

Signaling

The duration of the FS_X and FS_R pulses defines whether a frame is an information frame or a signaling frame:

- A frame synchronization pulse which is a full clock period in duration (CLK_X period for FS_X, CLK_R period for FS_R) designates a non-signaling frame.
- A frame synchronization pulse which is two full clock periods in duration (two CLK_X periods for FS_X, two CLK_R periods for FS_R) designates a signaling frame.

On the encoding side, when the FS_X pulse is widened, the 8th bit of the PCM word will be replaced by the value on the SIG_X input at the time when the 8th bit is output on the D_X lead.

On the decoding side, when the FS_R pulse is widened, the 8th bit of the PCM word is detected and transmitted on the SIG_R lead. That output is latched until the next receiving signaling frame.

The remaining 7 bits are decoded according to the value given in the CCITT G733 recommendation. The SIG_R lead is reset to a TTL low level whenever the Codec is in the power-down state.

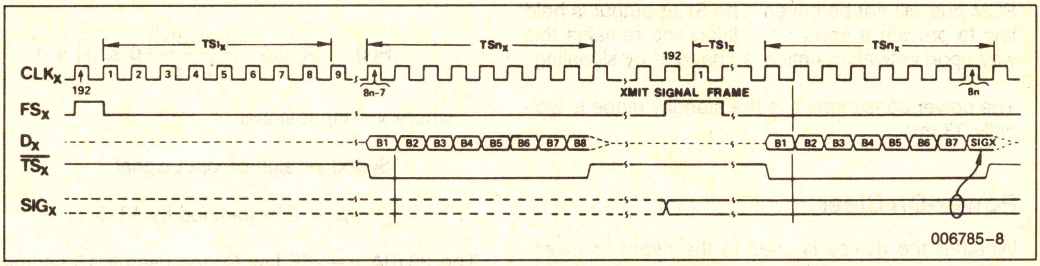


Figure 8. Transmit 8th Bit Signaling

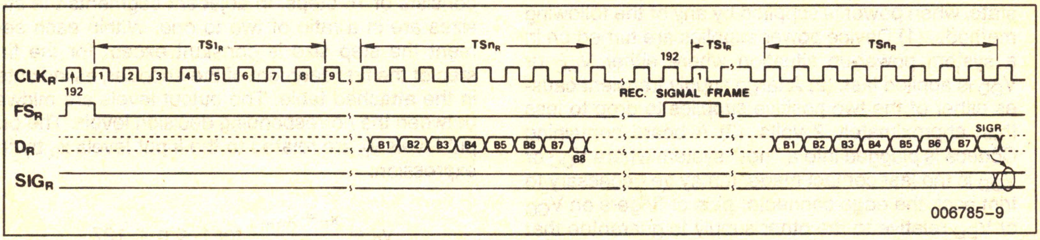


Figure 9. Receive 8th Bit Signaling

T1 Framing

The Codec will accept the standard D3/D4 framing format of 193 clock pulses per frame (equivalent to CLK_X, CLK_R of 1.544 Mb/s). However, the 193rd bit may be blanked (equivalent to CLK_X, CLK_R of 1.536 Mb/s) if desired.

Standby Mode—Power Down

To minimize power consumption and heat dissipation a standby mode is provided where all Codec functions are disabled except for D_C and CLK_C leads. These allow the Codec to be reactivated. In the microcomputer mode the Codec is placed into standby by loading a control word (D_C) with a "1" in bits 1 and 2 locations. In the direct mode when D_C is brought high, the all "1's" control word is internally transferred to the control register, invoking the standby condition.

While in the standby mode, the D_X output is actively held in a high impedance state to guarantee that the PCM bus will not be driven. The SIG_R output is held low to provide a known condition and remains this way upon activation until it is changed by signaling.

The power consumption in the standby mode is typically 33 mW.

Power-On Clear

Whether the device is used in the direct or microcomputer mode, an internal reset (power-on clear) is generated, forcing the device into the power down state, when power is supplied by any of the following methods. (1) Device power supplies are turned on in a system power-up situation where either V_{CC} or V_{DD} is applied last. (2) A large supply transient causes either of the two positive supplies to drop to less than approximately 2 volts. (3) A board containing Codecs is plugged into a "hot" system where V_{CC} or V_{DD} is the last contact made. It may be necessary to trim back the edge connector pins or fingers on V_{CC} or V_{DD} relative to the other supply to guarantee that the power-on clear will operate properly when a board is plugged into a "hot" system. Furthermore, the Codec will inhibit activity on $\overline{TS}_X$ and D_X during the application of power supplies.

The device is also tolerant of transients in the negative supply (V_{BB}) so long as V_{BB} remains more negative than −3.5 volts. V_{BB} transients which exceed this level should be detected and followed by a system reinitialization.

Precision Voltage Reference for the D/A Converter

The voltage reference is generated on the chip and is calibrated during the manufacturing process. The technique uses the difference in sub-surface charge density between two suitably implanted MOS devices to derive a temperature stable and bias stable reference voltage.

A gain setting op amp, programmed during manufacturing, "trims" the reference voltage source to the final precision voltage reference value provided to the D/A converter. The precision voltage reference determines the initial gain and dynamic range characteristics described in the A.C. Transmission Specification section.

μ-Law Conversion

μ-law represents a particular implementation of a piece-wise linear approximation to a logarithmic compression curve which is:

$$F(x) = \mathrm{Sgn}(x)\,\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \quad 0 \le |x| \le 1$$

where x = input signal

$\mathrm{Sgn}(x)$ = sign of input signal

μ = 255 (defined by AT & T)

The 2910A $\mu=255$ law Codec uses a 15 segment approximation to the logarithmic law. Each segment consists of 16 steps. In adjacent segments the step sizes are in a ratio of two to one. Within each segment the step size is constant except for the first step of the first segment of the encoder, as indicated in the attached table. The output levels are midway between the corresponding decision levels. The output levels y_n are related to the input levels x_n by the expression:

$$y_n = \frac{x_n + x_{n+1}}{2} \text{ for } 1 \le n \le 127$$

$$y_0 = x_0 = 0 \text{ for } n = 0$$

These relationships are implicit in the following table.

Theoretical μ-Law—Positive Input Values (for Negative Input Values, Invert Bit 1)

1	2	3	4	5	6	7	8
Segment Number	No. of Steps x Step Size	Value at Segment End Points	Decision Value Number n	Decision Value x_n[1]	PCM Word[3] MSB Bit Number LSB 1 2 3 4 5 6 7 8	Normalized Value at Decoder Output y_n[4]	Decoder Output Value Number
		8159[5]	(128)	(8159)			
					1 0 0 0 0 0 0 0	8031	127
8	16 x 256		127	7903	(see Note 2)		
			113	4319			
					1 0 0 0 1 1 1 1	4191	112
		4063	112	4063			
7	16 x 128				(see Note 2)		
			97	2143			
					1 0 0 1 1 1 1 1	2079	96
		2015	96	2015			
6	16 x 64				(see Note 2)		
			81	1055			
					1 0 1 0 1 1 1 1	1023	80
		991	80	991			
5	16 x 32				(see Note 2)		
			65	511			
					1 0 1 1 1 1 1 1	495	64
		479	64	479			
4	16 x 16				(see Note 2)		
			49	239			
					1 1 0 0 1 1 1 1	231	48
		223	48	223			
3	16 x 8				(see Note 2)		
			33	103			
					1 1 0 1 1 1 1 1	99	32
		95	32	95			
2	16 x 4				(see Note 2)		
			17	35			
					1 1 1 0 1 1 1 1	33	16
		31	16	31			
	15 x 2				(see Note 2)		
			2	3			
1 ↓					1 1 1 1 1 1 1 0	2	1
	1 x 1		1	1	1 1 1 1 1 1 1 1		
			0	0		0	0

NOTES:

1. 8159 normalized value units correspond to the value of the on-chip voltage reference.
2. The PCM word corresponding to positive input values between two successive decision values numbered n and n+1 (see column 4) is (255−n) expressed as a binary number.
3. The PCM word on the highways is the same as the one shown in column 6.
4. The voltage output on the VF_R lead is equal to the normalized value given in the table, augmented by an offset. The offset value is approximately 15 mV.
5. x_{128} is a virtual decision value.

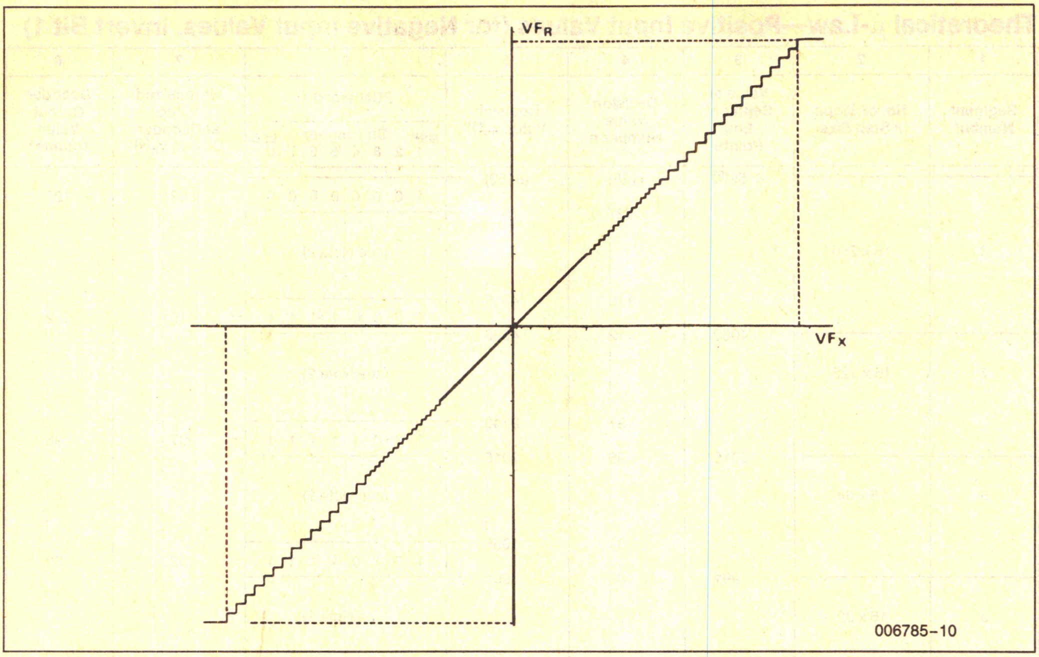


Figure 10. Codec Transfer Characteristic

During signaling frames, a 7-bit transfer characteristic is implemented in the decoder. This characteristic is derived from the decoder values in the attached table by assuming a value of "1" for the LSB (8th bit) and shifting the decoder transfer characteristics one half-step away from the origin. For example, the maximum decoder output level for signaling frames has normalized value 7903, whereas it has value 8031 in normal (non-signaling) frames.

APPLICATIONS

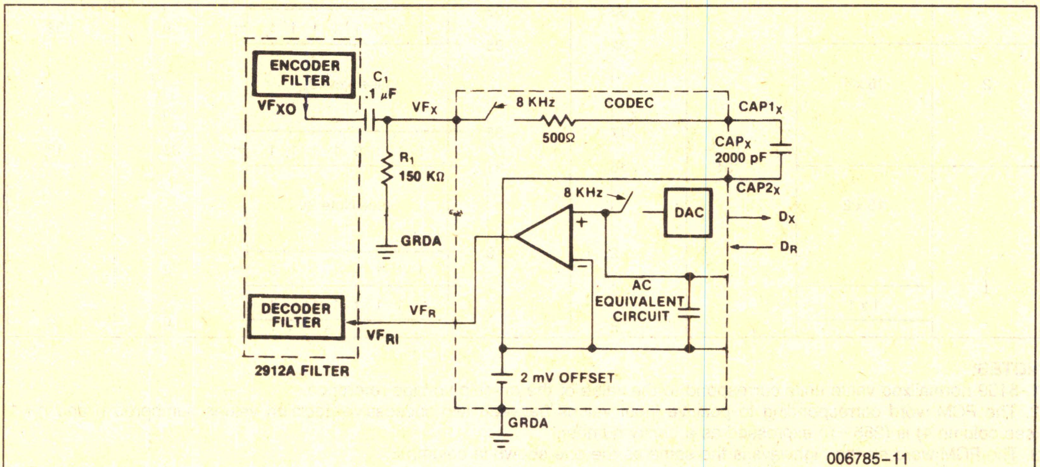


Figure 11. Circuit Interface—without External Auto Zero

Holding Capacitor

For an 8 KHz sampling system the transmit holding capacitor CAP_X should be 2000 pF ±20%.

Auto Zero

The 2910A contains a transparent on-chip auto zero plus a device pin for implementing a sign-bit driven external auto zero feedback loop. The on-chip auto zero reduces the input offset voltage of the encoder (VF_X) to less than 3 mV. For most telephony applications, this input offset is perfectly acceptable, since it insures the encoder is biased in the lower 25% of the first segment.

Where lower input offset is required the external auto zero loop may be used to bias the encoder exactly at the zero crossing point. The consequence of the external auto zero loop, aside from extra components, is the addition of the dithering auto-zero signal to the input signal, resulting in slightly higher idle channel noise (approximately 2dB) than when the external loop is not used. Consequently, where the application permits, it is recommended that the external auto zero loop not be used. When not used, the AUTO pin should float.

The circuit interface with auto zero drawing shows a possible connection between the VF_X and AUTO leads with the recommended values of $C_1 = 0.3\ \mu F$, $R_1 = 150\ K\Omega$, $R_2 = 330\ K\Omega$, and $R_3 = 470\ K\Omega$.

Filters Interface

The filters may be interfaced as shown in the circuit interface diagrams. Note that the output pulse stream is of the non-return-to-zero type and hence requires the (sin x)/x correction provided by the 2912A filter.

D_X Buffering

For higher drive capability or increased system reliability it may be desirable that the D_X output of a group of Codecs be buffered from the system PCM bus with an external three-state or open collector buffers. A buffer can be enabled with the appropriate Codec generated $\overline{TS}x$ signal or signals. $\overline{TS}x$ signal may also be used to activate external zero code suppression logic, since the occurrence of an active state of any $\overline{TS}x$ implies the existence of PCM voice bits (as opposed to transparent data bits) on the bus.

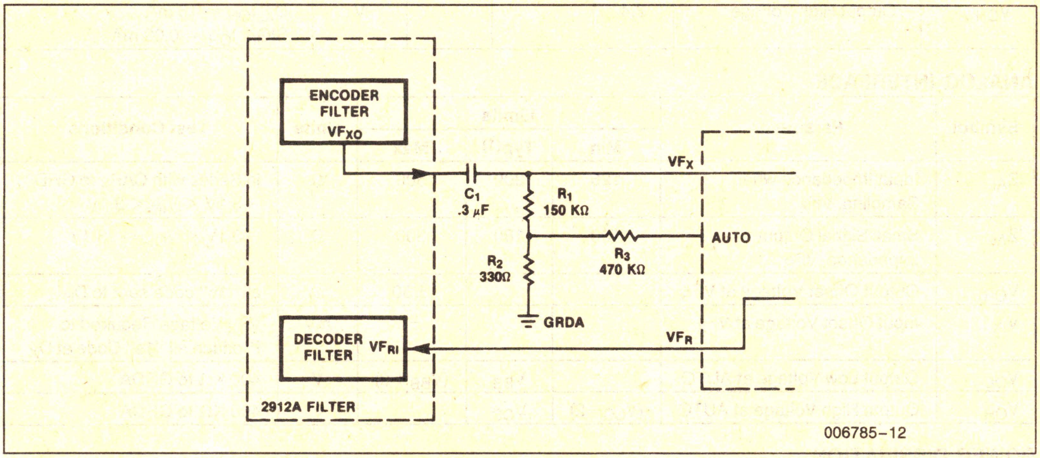


Figure 12. Circuit Interface—with External Auto Zero

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
All Input or Output Voltages with Respect to V_{BB} .... −0.3V to +20V
V_{CC}, V_{DD}, GRDD, and GRDA with Respect to V_{BB} .... −0.3V to +20V
Power Dissipation 1.35W

Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

$T_A = 0°C$ to $+70°C$, $V_{DD} = +12V$ ±5%, $V_{CC} = +5V$ ±5%, $V_{BB} = -5V$ ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified

DIGITAL INTERFACE

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ(1)	Max		
I_{IL}	Low Level Input Current			10	μA	$V_{IN} < V_{IL}$
I_{IH}	High Level Input Current			10	μA	$V_{IN} > V_{IH}$
V_{IL}	Input Low Voltage			0.6	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	D_X, $I_{OL} = 4.0$ mA SIG_R, $I_{OL} = 0.5$ mA $\overline{TSx}$, $I_{OL} = 3.2$ mA, Open Drain PDN, $I_{OL} = 1.6$ mA, Open Drain
V_{OH}	Output High Voltage	2.4			V	D_X, $I_{OH} = 15$ mA SIG_R, $I_{OH} = 0.08$ mA

ANALOG INTERFACE

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ(1)	Max		
Z_{AI}	Input Impedance when Sampling, VF_X	125	300	500	Ω	in Series with CAP_X to GRDA, $-3.1V < V_{IN} < 3.1V$
Z_{AO}	Small Signal Output Impedance, VF_R	100	180	300	Ω	$-3.1V < V_{OUT} < 3.1V$
V_{OR}	Output Offset Voltage at VF_R			±50	mV	all "1s" code sent to D_R
V_{IX}	Input Offset Voltage at VF_X			±5	mV	VF_X Voltage Required to Produce all "1s" Code at D_X
V_{OL}	Output Low Voltage at AUTO		V_{BB}	$(V_{BB}+2)$	V	400 KΩ to GRDA
V_{OH}	Output High Voltage at AUTO	$(V_{CC}-2)$	V_{CC}		V	400 KΩ to GRDA

POWER DISSIPATION

Symbol	Parameter	Limits			Units	Test Conditions
		Min	Typ(1)	Max		
I_{DDO}	Standby Current		0.7	1.1	mA	Auto Output = Open Clock Frequency = 2.048 MHz
I_{CCO}	Standby Current		4	7.0	mA	
I_{BBO}	Standby Current		1	2.5	mA	
I_{DDI}	Operating Current		11	16	mA	
I_{CCI}	Operating Current		13	21	mA	
I_{BBI}	Operating Current		4	6.0	mA	

NOTE:
1. Typical values are for $T_A = 25°C$ and nominal power supply values.

A.C. CHARACTERISTICS

$T_A = 0°C$ to $+70°C$, $V_{DD} = +12V \pm 5\%$, $V_{CC} = +5V \pm 5\%$, $V_{BB} = -5V \pm 5\%$, GRDA = 0V, GRDD = 0V, unless otherwise specified

TRANSMISSION

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
S/D	Signal/Tone Distortion Ratio, C-Message Weighted Half Channel (See Figure 1)	 36 30 27			 dB dB dB	VF_X = 1.02 KHz, Sinusoid $-30 \text{ dBm0} \leq VF_X \leq 0 \text{ dBm0}$ $-40 \text{ dBm0} \leq VF_X < -30 \text{ dBm0}$ $-45 \text{ dBm0} \leq VF_X < -40 \text{ dBm0}$
ΔG	 Gain Tracking Deviation Half Channel(2) Reference Level 0 dBm0		 ±0.25 ±0.60 ±1.5	 ±0.30 ±0.70 ±1.8	 dB dB dB	VF_X = 1.02 KHz, Sinusoid $-37 \text{ dBm0} \leq VF_X \leq +3 \text{ dBm0}$ $-50 \text{ dBm0} \leq VF_X < -37 \text{ dBm0}$ $-55 \text{ dBm0} \leq VF_X < -50 \text{ dBm0}$
ΔG_V	ΔG Variation with Supplies Half Channel		±0.0002 ±0.0004	±0.0004 ±0.0008	dB/mV dB/mV	$-37 \text{ dBm0} \leq VF_X \leq +3 \text{ dBm0}$ $-50 \text{ dBm0} \leq VF_X < -37 \text{ dBm0}$
ΔG_T	ΔG Variation with Temperature Half Channel		±0.001 ±0.002	±0.002 ±0.005	dB/°C dB/°C	$-37 \text{ dBm0} \leq VF_X \leq +3 \text{ dBm0}$ $-50 \text{ dBm0} \leq VF_X < -37 \text{ dBm0}$
N_{IC1}	Idle Channel Noise, C-Message Weighted		2	7	dBrnc0	No Signaling(3)
N_{IC2}	Idle Channel Noise, C-Message Weighted		10	13	dBrnc0	with 6th and 12th Frame Signaling(3)
N_{IC3}	Idle Channel Noise, C-Message Weighted		14	18	dBrnc0	with 1 KHz Sign Bit Toggle
HD	Harmonic Distortion (2nd or 3rd)		−48	−44	dB	VF_X = 1.02 KHz, 0 dBm0; Measured at Decoder Output VF_R
IMD	Intermodulation Distortion 2nd Order 3rd Order			 −45 −55	 dB dB	4-Tone Stimulus in Accordance with BSTR PUB 41009

NOTES:

1. Typical values are for T_A = 25°C and nominal supply values.
2. Measured in one direction, either decoder or encoder and an ideal device, at 23°C, nominal supplies.
3. If the external auto-zero is used N_{IC1} has a typical value of 8 dBrnc0 and N_{IC2} has a typical value of 13 dBrnc0.
4. D_R of Device Under Test (D.U.T.) driven with repetitive digital word sequence specified in CCITT recommendation G.711. Measurement made at VF_R output.
5. With the D.C. method the positive and negative clipping levels are measured and A_{IR} is calculated. With the A.C. method a sinusoidal input signal to VF_X is used where A_{IR} is measured directly.

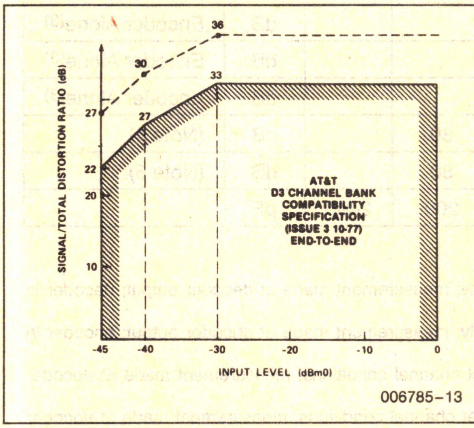


Figure 13. Signal/Total Distortion Ratio (Half-Channel)

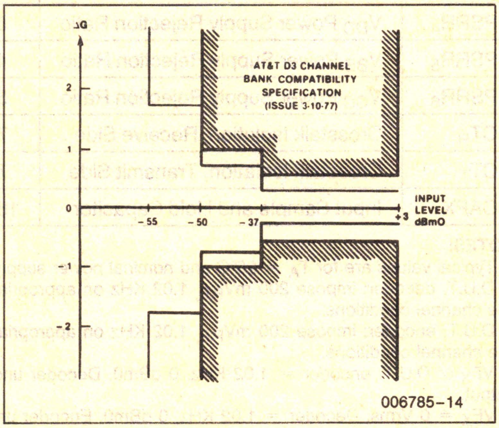


Figure 14. Gain Tracking Deviation (ΔG) (Half-Channel)

A.C. CHARACTERISTICS

T_A=0°C to +70°C, V_{DD}= +12V ±5%, V_{CC} = +5V ±5%, V_{BB}= −5V ±5%, GRDA=0V, GRDD=0V, unless otherwise specified (Continued)

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
DmW	Digital Milliwatt Response	5.53	5.63	5.73	dBm	23°C, Nominal Supplies(4)
DmW_T	DmW_O Variation with Temperature		−0.001	−0.002	dB/°C	Relative to 23°C(4)
DmW_S	DmW_O Variation with Supplies			±0.07	dB	Supplies ±5%(4)
A_{IR}	Input Dynamic Range	2.17	2.20	2.23	V_{RMS}	Using D.C. and A. C. Tests(5) 23°C, Nominal Supplies
A_{IRT}	Input Dynamic Range with Temperature			−0.5	mV_{RMS}/°C	Relative to 23°C
A_{IRS}	Input Dynamic Range with Supplies			±18	mV_{RMS}	Supplies ±5%
A_{OR}	Output Dynamic Range, VF_R	2.13	2.16	2.19	V_{RMS}	23°C, Nominal Supplies
A_{ORT}	A_{OR} Variation with Temperature			−0.5	mV_{RMS}/°C	Relative to 23°C
A_{ORS}	A_{OR} Variation with Supplies			±18	mV_{RMS}	Supplies ±5%

SUPPLY REJECTION AND CROSSTALK

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
$PSRR_1$	V_{DD} Power Supply Rejection Ratio	45			dB	Decoder Alone(2)
$PSRR_2$	V_{BB} Power Supply Rejection Ratio	35			dB	Decoder Alone(2)
$PSRR_3$	V_{CC} Power Supply Rejection Ratio	50			dB	Decoder Alone(2)
$PSRR_4$	V_{DD} Power Supply Rejection Ratio	50			dB	Encoder Alone(3)
$PSRR_5$	V_{BB} Power Supply Rejection Ratio	45			dB	Encoder Alone(3)
$PSRR_6$	V_{CC} Power Supply Rejection Ratio	50			dB	Encoder Alone(3)
CT_R	Crosstalk Isolation, Receive Side	75	80		dB	(Note 4)
CT_T	Crosstalk Isolation, Transmit Side	75	80		dB	(Note 5)
CAPX	Input Sample and Hold Capacitor	1600	200	2400	pF	

NOTES:
1. Typical values are for T_A = 25°C and nominal power supply values.
2. D.U.T. decoder; impose 200 $mV_{P.P.}$ 1.02 KHz on appropriate supply; measurement made at decoder output; decoder in idle channel conditions.
3. D.U.T. encoder; impose 200 $mV_{P.P.}$ 1.02 KHz on appropriate supply; measurement made at encoder output; encoder in idle channel conditions.
4. VF_X of D.U.T. encoder = 1.02 KHz, 0 dBm0. Decoder under quiet channel conditions; measurement made at decoder output.
5. VF_X = 0 Vrms. Decoder = 1.02 KHz, 0 dBm0. Encoder under quiet channel conditions; measurement made at encoder output.

A.C. CHARACTERISTIC—TIMING SPECIFICATION(1)

$T_A = 0°C$ to $+70°C$, $V_{DD} = +12V \pm 5\%$, $V_{CC} = +5V \pm 5\%$, $V_{BB} = -5V \pm 5\%$, GRDA = 0V, GRDD = 0V, unless otherwise specified

CLOCK SECTION

Symbol	Parameter	Limits Min	Limits Max	Units	Comments
t_{CY}	Clock Period	485		ns	CLK_X, CLK_R (2.048 MHz Systems), CLK_C
t_r, t_f	Clock Rise and Fall Time	0	30	ns	CLK_X, CLK_R, CLK_C
t_{CLK}	Clock Pulse Width	215		ns	CLK_X, CLK_R, CLK_C
t_{CDC}	Clock Duty Cycle ($t_{CLK} \div t_{CY}$)	45	55	%	CLK_X, CLK_R

TRANSMIT SECTION

Symbol	Parameter	Limits Min	Limits Max	Units	Comments
t_{VFX}	Analog Input Conversion	20		Timeslot	from Leading Edge of Transmit Timeslot (2)
t_{DZX}	Data Enabled on TS Entry	50	180	ns	$0 < C_{LOAD} < 100$ pF
t_{DHX}	Data Hold Time	80	230	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	75	245	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	30	220	ns	$0 < C_{LOAD} < 100$pF
t_{SOFF}	Timeslot X to Disable	70	225	ns	$C_{LOAD} = 0$
t_{SS}	Signal Setup Time	0		ns	Relative to Bit 7 Falling Edge
t_{SH}	Signal Hold Time	100		ns	Relative to Bit 8 Falling Edge
t_{FSD}	Frame Sync Delay	15	150	ns	

RECEIVE AND CONTROL SECTIONS

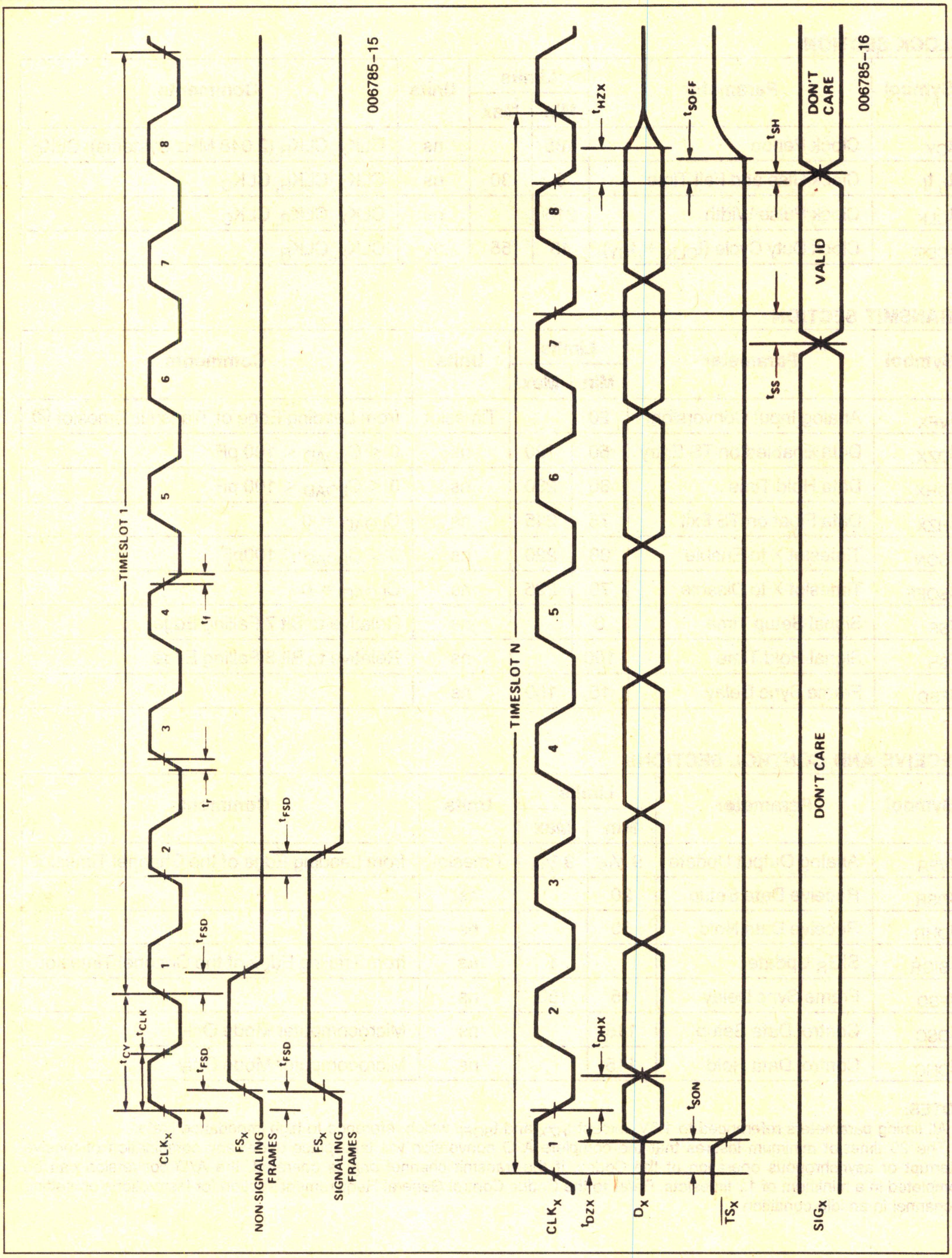
Symbol	Parameter	Limits Min	Limits Max	Units	Comments
t_{VFR}	Analog Output Update	$9\frac{1}{16}$	$9\frac{1}{16}$	Timeslot	from Leading Edge of the Channel Timeslot
t_{DSR}	Receive Data Setup	20		ns	
t_{DHR}	Receive Data Hold	60		ns	
t_{SIGR}	SIG_R Update		1	μs	from Trailing Edge of the Channel Timeslot
t_{FSD}	Frame Sync Delay	15	150	ns	
t_{DSC}	Control Data Setup	115		ns	Microcomputer Mode Only
t_{DHC}	Control Data Hold	115		ns	Microcomputer Mode Only

NOTES:

1. All timing parameters referenced to 1.5V, except t_{HZX} and t_{SOFF} which reference to high impedance state.
2. The 20 timeslot minimum insures that the complete A/D conversion will take place under any combination of receive interrupt or asynchronous operation of the Codec. If the transmit channel *only* is operated, the A/D conversion can be completed in a minimum of 11 timeslots. Refer to the Codec Control General Requirement section for instructions on setting a channel in an idle condition.

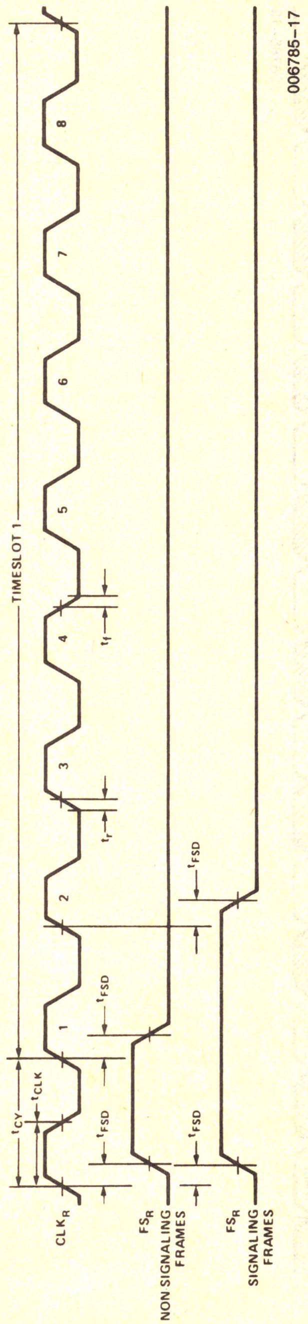
TIMING WAVEFORMS

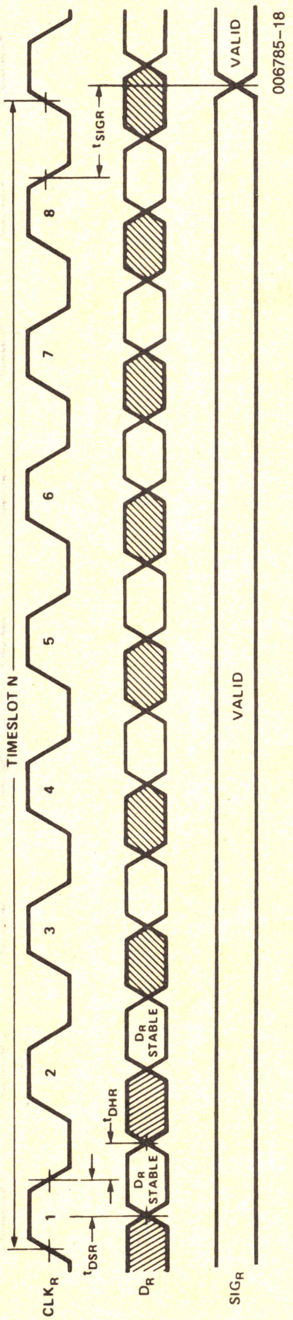
TRANSMIT TIMING



TIMING WAVEFORMS (Continued)

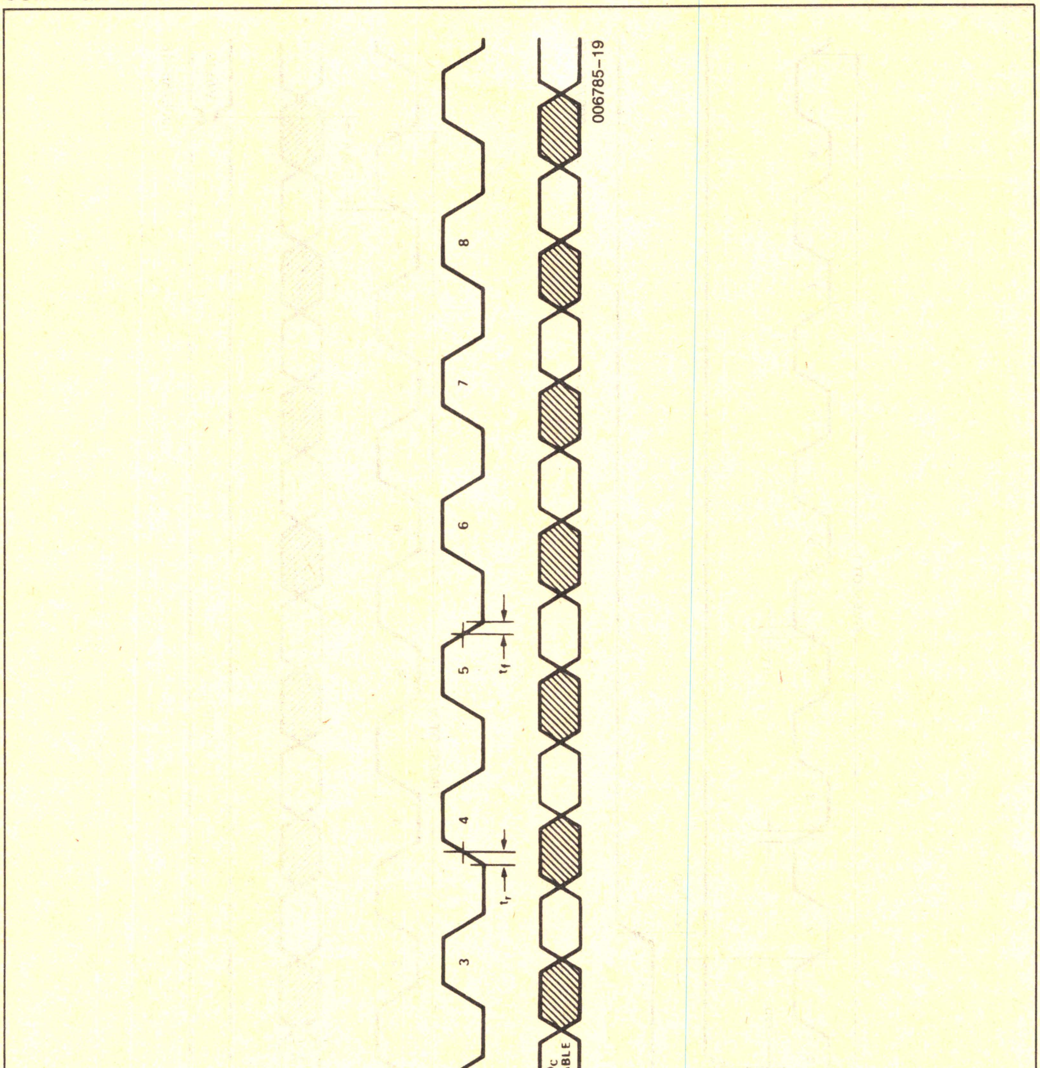
RECEIVE TIMING





TIMING WAVEFORMS (Continued)

CONTROL TIMING



2911A-1
PCM CODEC—A LAW
8-BIT COMPANDED A/D AND D/A CONVERTER

- **Per Channel, Single Chip Codec**
- **CCITT G711 and G732 Compatible, Even Order Bits Inversion Included**
- **Microcomputer Interface with On-Chip Time-Slot Computation**
- **Simple Direct Mode Interface When Fixed Timeslots Are Used**
- **±5% Power Supplies: +12V, +5V, −5V**
- **66 dB Dynamic Range, with Resolution Equivalent to 11-Bit Linear Conversion Around Zero**
- **Precision On-Chip Voltage Reference**
- **Low Power Consumption 230 mW Typ. Standby Power 33 mW Typ.**
- **Fabricated with Reliable N-Channel MOS Process**

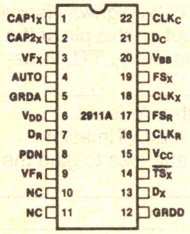
The Intel 2911A is a fully integrated PCM (Pulse Code Modulation) Codec (Coder-Decoder), fabricated with N-channel silicon gate technology. The high density of integration allows the sample and hold circuits, the digital-to-analog converter, the comparator and the successive approximation register to be integrated on the same chip, along with the logic necessary to interface a full duplex PCM link.

The primary applications are in telephone systems:

- Transmission — 30/32 Channel Systems at 2.048 Mbps
- Switching — Digital PBX's and Central Office Switching Systems
- Concentration — Subscriber Carrier/Concentrators

The wide dynamic range of the 2911A (66 dB) and the minimal conversion time (80 μs minimum) make it an ideal product for other applications, like:

- Data Acquisition
- Telemetry
- Secure Communications Systems
- Signal Processing Systems

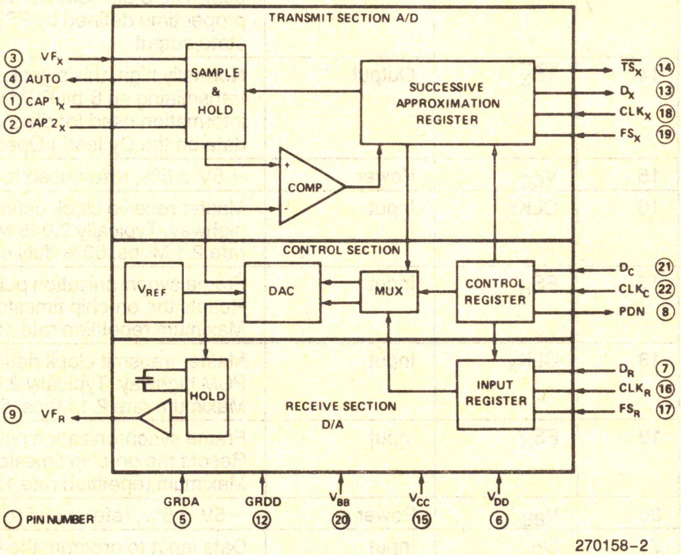


270158–1

Figure 1. Pin Configuration

CAP 1_X, CAP 2_X	Holding Capacitor
VF_X	Analog Input
VF_R	Analog Output
D_R, D_C	Digital Input
D_X, $\overline{TS_X}$	Digital Output
CLK_C, CLK_X, CLK_R	Clock Input
FS_X, FS_R	Frame Sync Input
AUTO	Auto Zero Output
V_{BB}	Power (−5V)
V_{CC}	Power (+5V)
V_{DD}	Power (+12V)
PDN	Power Down
GRDA	Analog Ground
GRDD	Digital Ground
NC	No Connect

Figure 2. Pin Names



270158–2

Figure 3. Block Diagram

November 1986
Order Number: 270158-001

PIN DESCRIPTION

Pin No	Symbol	Function	Description
1	$CAP1_X$	Hold	Connections for the transmit holding capacitor. Refer to Applications section.
2	$CAP2_X$		
3	VF_X	Input	Analog input to be encoded into a PCM word. The signal on this lead is sampled at the same rate as the transmit frame synchronization pulse FS_X, and the sample value is held in the external capacitor connected to the $CAP1_X$ and $CAP2_X$ leads until the encoding process is completed.
4	AUTO	Output	Most significant bit of the encoded PCM word (+5V for negative, −5V for positive values). Refer to the Codec Applications section.
5	GRDA	Ground	Analog return common to the transmit and receive analog circuits. Not connected to GRDD internally.
6	V_{DD}	Power	+12V ±5%; referenced to GRDA.
7	D_R	Input	Receive PCM highway (serial bus) interface. The Codec serially receives a PCM word (8 bits) through this lead at the proper time defined by FS_R, CLK_R, D_C, and CLK_C.
8	PDN	Output	Active high when the Codec is in the power down state. Open drain output.
9	VF_R	Output	Analog Output. The voltage present on VF_R is the decoded value of the PCM word received on lead D_R. This value is held constant between two conversions.
10	NC	No Connects	Recommended practice is to strap these NC's to GRDA.
11	NC		
12	GRDD	Ground	Ground return common to the logic power supply; V_{CC}.
13	D_X	Output	Output of the transmit side onto the send PCM highway (serial bus). The 8-bit PCM word is serially sent out on this pin at the proper time defined by FS_X, CLK_X, D_C, and CLK_C. TTL three-state output.
14	$\overline{TS_X}$	Output	Normally high, this signal goes low while the Codec is transmitting an 8-bit PCM word on the D_X lead. (Timeslot information used for diagnostic purposes and also to gate the data on the D_X lead.) Open drain output.
15	V_{CC}	Power	+5V ±5%, referenced to GRDD.
16	CLK_R	Input	Master receive clock defining the bit rate on the receive PCM highway. Typically 2.048 Mbps for a carrier system. Maximum rate 2.1 Mbps. 50% duty cycle. TTL compatible.
17	FS_R	Input	Frame synchronization pulse for the receive PCM highway. Resets the on-chip timeslot counter for the receive side. Maximum repetition rate 12 KHz. TTL interface.
18	CLK_X	Input	Master transmit clock defining the bit rate on the transmit PCM highway. Typically 2.048 Mbps for a carrier system. Maximum rate 2.1 Mbps. 50% duty cycle. TTL interface.
19	FS_X	Input	Frame synchronization pulse for the transmit PCM highway. Resets the on-chip timeslot counter for the transmit side. Maximum repetition rate 12 KHz. TTL interface.
20	V_{BB}	Power	−5V ±5%, referenced to GRDA.
21	D_C	Input	Data input to program the Codec for the chosen mode of operation. Becomes an active low chip select when CLK_C is tied to V_{CC}. TTL interface.
22	CLK_C	Input	Clock input to clock in the data on the D_C lead when the timeslot assignment feature is used; tied to V_{CC} to disable this feature. TTL interface.

FUNCTIONAL DESCRIPTION

The 2911A PCM Codec provides the analog-to-digital and the digital-to-analog conversions necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. The Codec is intended to be used on line and trunk terminations.

In a typical telephone system the Codec is located between the PCM highways and the channel filters.

The Codec encodes the incoming analog signal at the frame rate (FS_X) into an 8-bit PCM word which is sent out on the D_X lead at the proper time. Similarly, on the receive link, the Codec fetches an 8-bit PCM word from the receive highway (D_R lead) and decodes an analog value which will remain constant on lead VF_R until the next receive frame. Transmit and receive frames are independent. They can be asynchronous (transmission) or synchronous (switching) with each other.

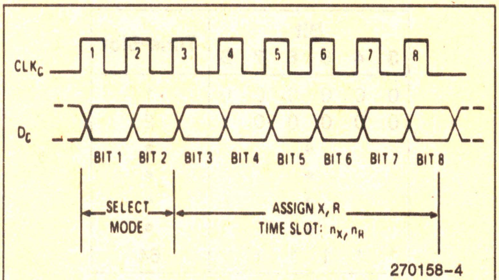
Circuitry is provided within the Codec to internally define the transmit and receive timeslots. In small systems this may eliminate the need for any external timeslot exchange; in large systems it provides one level of concentration. This feature can be bypassed and discrete timeslots sent to each Codec within a system.

In the power-down mode, most functions of the Codec are directly disabled to reduce power dissipation to a minimum.

CODEC OPERATION

Codec Control

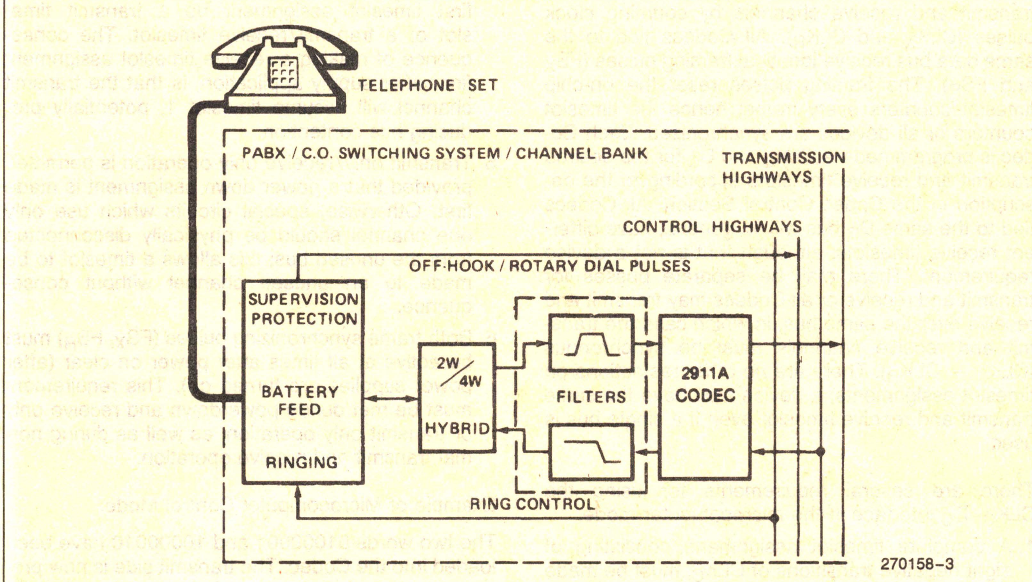
The operation of the 2911A is defined by serially loading an 8-bit word through the D_C lead (data) and the CLK_C lead (clock). The loading is synchronous with the other operations of the Codec, and takes place whenever transitions occur on the CLK_C lead. The D_C input is loaded in during the trailing edge of the CLK_C input.



270158-4

The control word contains two fields:

Bit 1 and Bit 2 define whether the subsequent 6 bits apply to both the transmit and receive side (00), the transmit side only (01), the receive side only (10), or whether the Codec should go into the standby, power-down mode (11). In the last case (11), the following 6 bits are irrelevant.



270158-3

Figure 4. Typical Line Termination

The last 6 bits of the control word define the timeslot assignment, from 000000 (timeslot 1) to 111111 (timeslot 64). Bit 3 is the most significant bit and bit 8 the least significant bit and last into the Codec.

Bit 1	Bit 2	Mode
0	0	X & R
0	1	X
1	0	R
1	1	Standby

Bit 3	4	5	6	7	8	Time-Slot
0	0	0	0	0	0	1
0	0	0	0	0	1	2
		•				•
		•				•
		•				•
		•				•
1	1	1	1	1	1	64

The Codec will retain the control word (or words) until a new word is loaded in or until power is lost. This feature permits dynamic allocation of timeslots for switching applications.

Microcomputer Control Mode

In the microcomputer mode, each Codec performs its own timeslot computation independently for the transmit and receive channels by counting clock pulses (CLK_X and CLK_R). All Codecs tied to the same data bus receive identical framing pulses (FS_X and FS_R). The framing pulses reset the on-chip timeslot counters every frame; hence the timeslot counters of all devices are synchronized. Each Codec is programmed via CLK_C and D_C for the desired transmit and receive timeslots according to the description in the Codec Control Section. All Codecs tied to the same D_R bus will, in general, have different receive timeslots, although that is not a device requirement. There may be separate busses for transmit and receive or all Codecs may transmit and receive over the same bus, in which case the transmit and receive channels must be synchronous ($CLK_X = CLK_R$). There are no other restrictions on timeslot assignments; a device may have the same transmit and receive timeslot even if a single bus is used.

There are several requirements for using the CLK_C–D_C interface in the microcomputer mode.

1. A complete timeslot assignment, consisting of eight negative transitions of CLK_C, must be made in less than one frame period. The assignment can overlap a framing pulse so long as all 8 control bits are clocked in within a total span of 125 μs (for an 8 KHz frame rate). CLK_C must be left at a TTL low level when not assigning a timeslot.
2. A dead period of two frames must always be observed between successive timeslot assignments. The two frame delay is measured from the rising edge of the first CLK_C transition of the previous timeslot assigned.
3. When the device is in the power-down state (Standby), the following three-step sequence must be followed to power-up the codec to avoid contention on the transmit PCM highway.
   a. Assign a dummy transmit timeslot. The dummy should be at least two timeslots greater than the maximum valid system (usually 24 or 32). For example, in a 24 timeslot system, the dummy could be any timeslot between 26 and 64. This will power-up the transmit side, but prevent any spurious Dx or $\overline{TSx}$ outputs.
   b. Two frames later, assign the desired transmit timeslot.
   c. Two frames later assign the desired receive timeslot.
4. Initialization sequence: The device contains an on-chip power-on clear function which guarantees that with proper sequencing of the supplies (V_{CC} or V_{DD} on last), the device will initialize with no timeslot assigned to either the transmit or receive channel. After a supply failure or whenever the supplies are applied, it is recommended that either power down assignment be made first, or the first timeslot assignment be a transmit timeslot or a transmit/receive timeslot. The consequence of making a receive timeslot assignment first, after supply application, is that the transmit channel will assume timeslot 1, potentially producing bus contention.
5. Transmit only/receive only operation is permitted provided that a power down assignment is made first. Otherwise, special circuits which use only one channel should be physically disconnected from the unused bus; this allows a timeslot to be made to an unused channel without consequence.
6. Both frame synchronizing pulses (FS_X, FS_R) must be active at all times after power on clear (after power supplies are turned on). This requirement must be met during powerdown and receive only or transmit only operation, as well as during normal transmit and receive operation.

Example of Microcomputer Control Mode:

The two words 01000001 and 10000010 have been loaded into the Codec. The transmit side is now programmed for timeslot 2 and the receive side for

timeslot 3. The Codec will output a PCM word on the transmit PCM highway (bus) during the timeslot 2 of the transmit frame, and will fetch a PCM word from the receive PCM highway during timeslot 3.

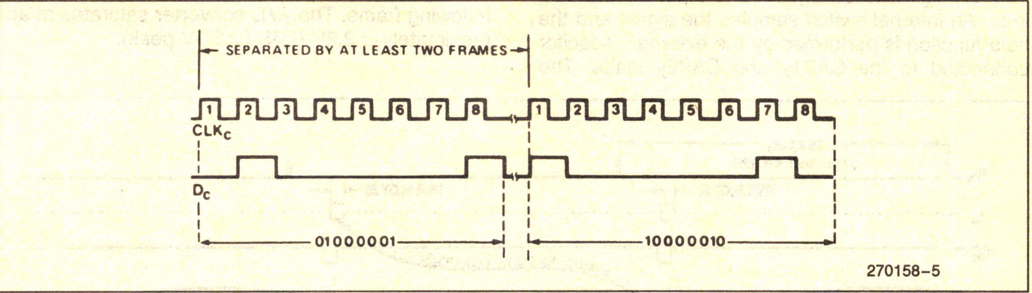


Figure 5. Microcomputer Mode Programming Examples

In this example the Codec interface to the PCM highway then functions as shown below. (FS_X and FS_R may be asynchronous.)

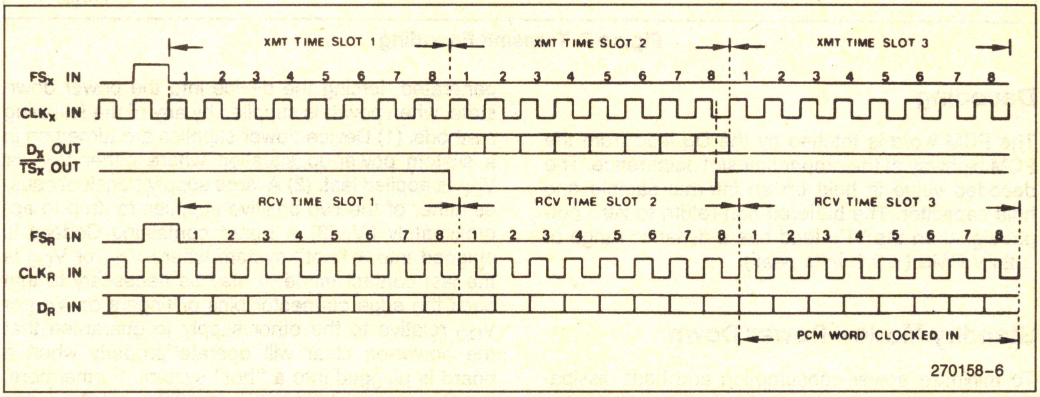


Figure 6. Microcomputer Mode PCM Highway Example

Direct Control Mode

The direct mode of operation will be selected when the CLK_C pin is strapped to the +5V supply (V_{CC}). In this mode, the D_C pin is an active low chip select. In other words, when D_C is low, the device transmits and receives in the timeslots which follow the appropriate framing pulses. With D_C high the device is in the power down state. Even though CLK_C characteristics are simpler for the 2911A it will operate properly when plugged into a 2911 board.

Deactivation of a channel by removal of the appropriate framing pulse (FS_X or FS_R) is not permitted.

Specifically, framing pulses must be applied for a minimum of two frames after a change in state of D_C in order for the D_C change to be internally sensed. In particular, when entering standby in the direct mode, framing pulses must be applied as usual for two frames after D_C is brought high.

The Codec will enter direct mode within three frame times (375 μs) as measured from the time the device power supplies settle to within the specified limits. This assumes that CLK_C is tied to V_{CC} and that all clocks are available at the time the supplies have settled.

General Control Requirements

All bit and frame clocks should be applied whenever the device is active. In particular, an unused channel cannot be deactivated by removal of its associated frame or bit clock while the other channel of the same device remains active.

A single channel cannot be deactivated except by physical disconnection of the data lead (D_X or D_R) from the system data bus. A device (both transmit and receive channels) may be deactivated in either control mode by powering down the device. Both channels are always powered down together.

Encoding

The VF signal to be encoded is input on the VF_X lead. An internal switch samples the signal and the hold function is performed by the external capacitor connected to the $CAP1_X$ and $CAP2_X$ leads. The sampling and conversion is synchronized with the transmit timeslot. The PCM word is then output on the D_X lead at the proper timeslot occurrence of the following frame. The A/D converter saturates at approximately ±2.2V RMS (±3.1V peak).

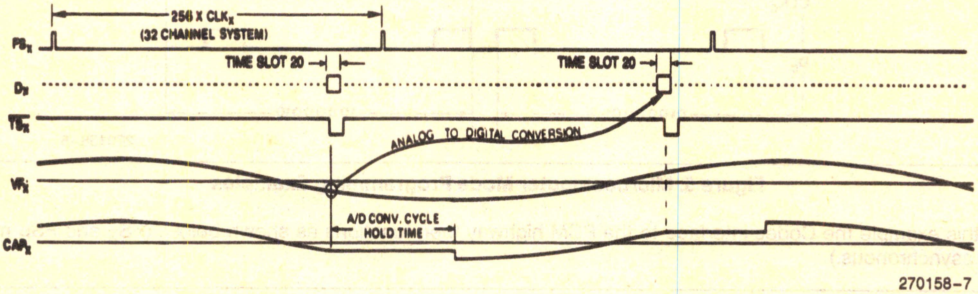


Figure 7. Transmit Encoding

Decoding

The PCM word is fetched by the D_R lead from the PCM highway at the proper timeslot occurrence. The decoded value is held on an internal sample and hold capacitor. The buffered non-return to zero output signal on the VF_R lead has a dynamic range of ±2.2V RMS (±3.1 volts peak).

Standby Mode—Power Down

To minimize power consumption and heat dissipation a standby mode is provided where all Codec functions are disabled except for D_C and CLK_C leads. These allow the Codec to be reactivated. In the microcomputer mode the Codec is placed into standby by loading a control word (D_C) with a "1" in bits 1 and 2 locations. In the direct mode when D_C is brought high, the all "1's" control word is internally transferred to the control register, invoking the standby condition.

While in the standby mode, the D_X output is actively held in a high impedance state to guarantee that the PCM bus will not be driven.

The power consumption in the standby mode is typically 33 mW.

Power-On Clear

Whether the device is used in the direct or microcomputer mode, an internal reset (power-on clear) is generated, forcing the device into the power down state, when power is supplied by any of the following methods. (1) Device power supplies are turned on in a system power-up situation where either V_{CC} or V_{DD} is applied last. (2) A large supply transient causes either of the two positive supplies to drop to approximately 2V. (3) A board containing Codecs is plugged into a "hot" system where V_{CC} or V_{DD} is the last contact made. It may be necessary to trim back the edge connector pins or fingers on V_{CC} or V_{DD} relative to the other supply to guarantee that the power-on clear will operate properly when a board is plugged into a "hot" system. Furthermore, the Codec will inhibit activity on $\overline{TS}_X$ and D_X during the application of power supplies.

The device is also tolerant of transients in the negative supply (V_{BB}) so long as V_{BB} remains more negative than −3.5V. V_{BB} transients which exceed this level should be detected and followed by a system reinitialization.

Precision Voltage Reference for the D/A Converter

The voltage reference is generated on the chip and is calibrated during the manufacturing process. The technique uses the difference in sub-surface charge density between two suitably implanted MOS devices to derive a temperature stable and bias stable reference voltage.

A gain setting op amp, programmed during manufacturing, "trims" the reference voltage source to the final precision voltage reference value provided to the D/A converter. The precision voltage reference determines the initial gain and dynamic range characteristics described in the A.C. Transmission Specification section.

CONVERSION LAW

The conversion law is commonly referred to as the A Law.

$$F(x) = \mathrm{Sgn}(x)\left[\frac{1 + \log_{10}(A|X|)}{1 + \log_{10} A}\right], \quad 1/A \leq |x| \leq 1$$

$$F(x) = \mathrm{Sgn}(x)\left[\frac{A|X|}{1 + \log_{10} A}\right], \quad 0 \leq |x| \leq 1/A$$

where: x = the input signal

Sgn(x) = sign of the input signal

A = 87.6 (defined by CCITT)

The Codec provides a piecewise linear approximation of the logarithmic law through 13 segments. Each segment is made of 16 steps with the exception of the first segment, which has 32 steps. In adjacent segments the step sizes are in a ratio of two to one. Within each segment, the step size is constant.

The output levels are midway between the corresponding decision levels. The output levels y_n are related to the input levels x_n by the expression:

$$y_n = \frac{x_{n-1} + x_n}{2}, \quad 0 < n \leq 128$$

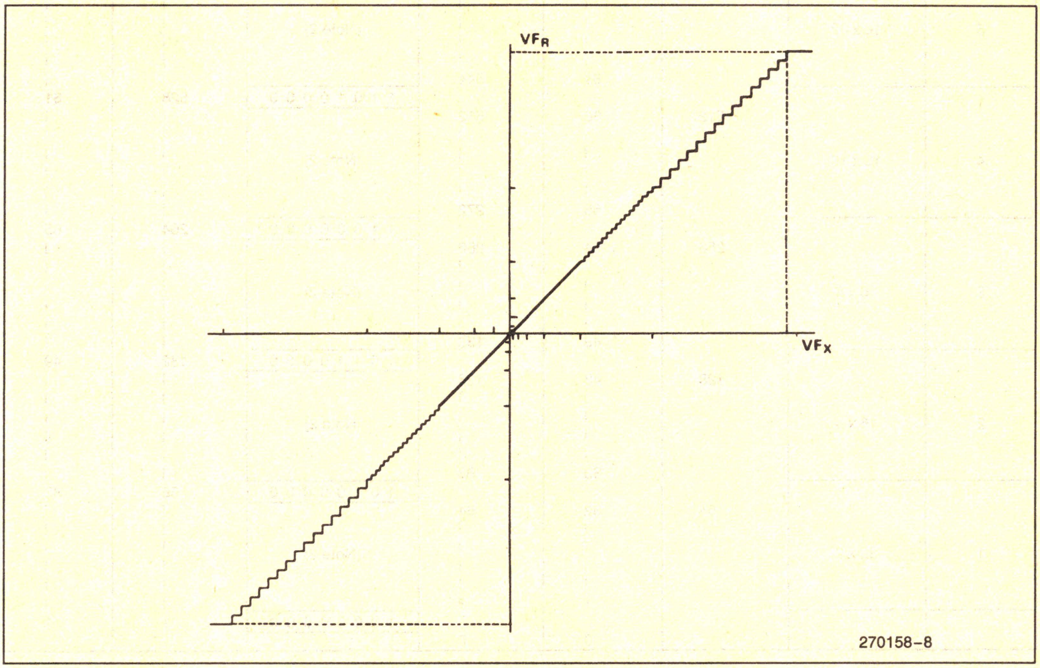


Figure 10. Codec Transfer Characteristic

Theoretical A-Law—Positive Input Values (for Negative Input Values, Invert Bit 1)

1	2	3	4	5	6	7	8
Segment Number	No. of Steps x Step Size	Value at Segment End Points	Decision Value Number n	Decision Value x_n(1)	PCM Word(4) Bit Number 1 2 3 4 5 6 7 8	Normalized Value at Decoder Output y_n(5)	Decoder Output Value Number
		4096(3)	(128)	(4096)			
					1 1 1 1 1 1 1 1	4032	128
			127	3968			
7	16 x 128				(Note 2)		
			113	2176			
					1 1 1 1 0 0 0 0	2112	113
		2048	112	2048			
6	16 x 64				(Note 2)		
			97	1088			
					1 1 1 0 0 0 0 0	1056	97
		1024	96	1024			
5	16 x 32				(Note 2)		
			81	544			
					1 1 0 1 0 0 0 0	528	81
		512	80	512			
4	16 x 16				(Note 2)		
			65	272			
					1 1 0 0 0 0 0 0	264	65
		256	64	256			
3	16 x 8				(Note 2)		
			49	136			
					1 0 1 1 0 0 0 0	132	49
		128	48	128			
2	16 x 4				(Note 2)		
			33	68			
					1 0 1 0 0 0 0 0	66	33
		64	32	64			
1	32 x 2				(Note 2)		
			1	2			
					1 0 0 0 0 0 0 0	1	1
			0	0			

NOTES:

1. 4096 normalized value units correspond to the value of the on-chip voltage reference.
2. The PCM word corresponding to positive input values between two successive decision values numbered n and n + 1 (see column 4) is (128 + n) expressed as a binary number.
3. X_{128} is a virtual decision value.
4. The PCM word on the highways is the same as the one shown in column 6, with the even order bits inverted. The 2911A provides for the inversion of the even order bits on both the send and receive sections.
5. The voltage output on the VF_R lead is equal to the normalized value given in the table, augmented by an offset. The offset value is approximately 15 mV.

APPLICATIONS

Holding Capacitor

For an 8 KHz sampling system the transmit holding capacitor CAP_X should be 2000 pF ±20%.

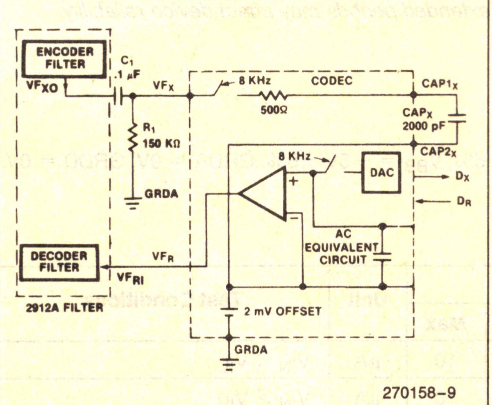


Figure 11. Circuit Interface—Without External Auto Zero

Filters Interface

The filters may be interfaced as shown in the circuit interface diagrams. Note that the output pulse stream is of the non-return-to-zero type and hence requires the (sin x)/x correction provided by the 2912A filter.

D_X Buffering

For higher drive capability or increased system reliability it may be desirable that the D_X output of a group of Codecs be buffered from the system PCM bus with an external three-state or open collector buffers. A buffer can be enabled with the appropriate Codec generated $\overline{TS}_X$ signal or signals. $\overline{TS}_X$ signal may also be used to activate external zero code suppression logic, since the occurrence of an active state of any $\overline{TS}_X$ implies the existence of PCM voice bits (as opposed to transparent data bits) on the bus.

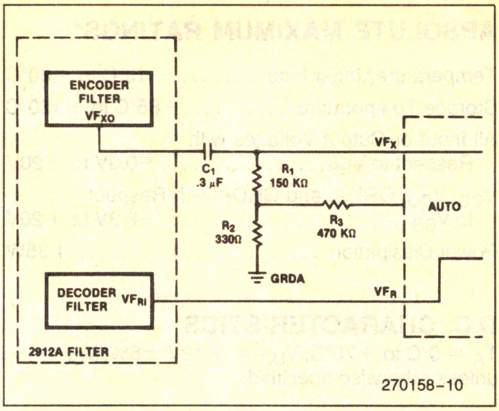


Figure 12. Circuit Interface—With External Auto Zero

Auto Zero

The 2911A contains a transparent on-chip auto zero plus a device pin for implementing a sign-bit driven external auto zero feedback loop. The on-chip auto zero reduces the input offset voltage of the encoder (VF_X) to less than 3 mV. For most telephony applications, this input offset is perfectly acceptable, since it insures the encoder is biased in the lower 25% of the first segment.

Where lower input offset is required the external auto zero loop may be used to bias the encoder exactly at the zero crossing point. The consequence of the external auto zero loop, aside from extra components, is the addition of the dithering auto-zero signal to the input signal, resulting in slightly higher idle channel noise (approximately 2 dB) than when the external loop is not used. Consequently, where the application permits, it is recommended that the external auto zero loop not be used. When not used, the AUTO pin should float.

The circuit interface with external auto zero drawing shows a possible connection between VF_X and AUTO leads with the recommended values of $C_1 = 0.3$ μF, $R_1 = 150$ KΩ, $R_2 = 330$Ω, and $R_3 = 470$ KΩ.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
All Input or Output Voltages with Respect to V_{BB} .... −0.3V to +20V
V_{CC}, V_{DD}, GRDA, and GRDA with Respect to V_{BB} .... −0.3V to +20V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified.

DIGITAL INTERFACE

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
I_{IL}	Low Level Input Current			10	μA	$V_{IN} < V_{IL}$
I_{IH}	High Level Input Current			10	μA	$V_{IN} > V_{IH}$
V_{IL}	Input Low Voltage			0.6	V	
V_{IH}	Input High Voltage	2.2			V	
V_{OL}	Output Low Voltage			0.4	V	D_X, I_{OL} = 4.0 mA $\overline{TS}_X$, I_{OL} = 3.2 mA, open drain PDN, I_{OL} = 1.6 mA, open drain
V_{OH}	Output High Voltage	2.4			V	D_X, I_{OH} = 15 mA

ANALOG INTERFACE

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
Z_{AI}	Input Impedance when Sampling, VF_X	125	300	500	Ω	In series with CAP_X to GRDA, $-3.1V < V_{IN} < 3.1V$
Z_{AO}	Small Signal Output Impedance, VF_R	100	180	300	Ω	$-3.1V < V_{OUT} < 3.1V$
V_{OR}	Output Offset Voltage at VF_R	−50		50	mV	Minimum code to D_R
V_{IX}	Input Offset Voltage at VF_X	−5		5	mV	Minimum positive code produced at D_X
V_{OL}	Output Low Voltage at AUTO		V_{BB}	$(V_{BB}+2)$	V	400 KΩ to GRDA
V_{OH}	Output High Voltage at AUTO	$(V_{CC}-2)$	V_{CC}		V	400 KΩ to GRDA

D.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified. (Continued)

POWER DISSIPATION

Symbol	Parameter	Limits Min	Limits Typ(1)	Limits Max	Unit	Test Conditions
I_{DDO}	Standby Current		0.7	1.1	mA	Auto Output = Open Clock Frequency = 2.048 MHZ
I_{CCO}	Standby Current		4.0	7.0	mA	
I_{BBO}	Standby Current		1.0	2.5	mA	
I_{DDI}	Operating Current		11	16	mA	
I_{CCI}	Operating Current		13	21	mA	
I_{BBI}	Operating Current		4.0	6.0	mA	

NOTE:
1. Typical values are for T_A = 25°C and nominal power supply values.

A.C. CHARACTERISTICS

T_A = 0°C to +70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified.

TRANSMISSION

Symbol	Parameter	Limits Min	Limits Typ(1)	Limits Max	Unit	Test Conditions
S/D	Signal to Total Distortion Ratio. CCITT G.712 Method 2 (Half Channel)	37			dB	Signal level 0 dBm0 to −30 dBm0
		31			dB	Signal level to −40 dBm0
		26			dB	Signal level to −45 dBm0
ΔG	2911A Gain Tracking Deviation Half Channel(3) Reference Level −10 dBm0		 ±0.25 ±0.60 ±1.5	 ±0.30 ±0.70 ±1.8	 dB dB dB	VF_X = 1.02 KHz, sinusoid −40 dBm0 ≤ VF_X ≤ +3 dBm0 −50 dBm0 ≤ VF_X < −40 dBm0 −55 dBm0 ≤ VF_X < −50 dBm0
ΔG_V	ΔG Variation with Supplies Half Channel		±0.0002 ±0.0004	±0.0004 ±0.0008	dB/mV dB/mV	−40 dBm0 ≤ VF_X ≤ +3 dBm0 −50 dBm0 ≤ VF_X < −40 dBm0
ΔG_T	ΔG Variation with Temperature Half Channel		±0.001 ±0.002	±0.002 ±0.005	dB/°C dB/°C	−40 dBm0 ≤ VF_X ≤ +3 dBm0 −50 dBm0 ≤ VF_X < −40 dBm0
N_{IC}	Idle Channel Noise		−85	−78	dBm0p	Quiet Code. (Note 2)
HD	Harmonic Distortion (2nd or 3rd)		−48	−44	dB	VF_X = 1.02 KHz, 0 dBm0; measured at decoder output VF_R
 IMD_1 IMD_2	Intermodulation Distortion G.712(7.1) G.712(7.2)			 −45 −50	 dB dBm0	 CCITT G.712 Two Tone Method

A.C. CHARACTERISTICS

$T_A = 0°C$ to $+70°C$, $V_{DD} = +12V \pm 5\%$, $V_{CC} = +5V \pm 5\%$, $V_{BB} = -5V \pm 5\%$, GRDA = 0V, GRDD = 0V, unless otherwise specified. (Continued)

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
DmW	Digital Milliwatt Response	5.58	5.66	5.78	dBm	23°C, nominal supplies(4)
DmW_T	DmW_O Variation with Temperature		−0.001	−0.002	dB/°C	Relative to 23°C(4)
DmW_S	DmW_O Variation with Supplies			±0.07	dB	Supplies ±5%(4)
A_{IR}	Input Dynamic Range	2.183	2.213	2.243	V_{RMS}	Using D.C. and A.C. tests(5) 23°C, nominal supplies
A_{IRT}	Input Dynamic Range vs Temperature			−0.5	mV_{RMS}/°C	Relative to 23°C
A_{IRS}	Input Dynamic Range vs Supplies			±18	mV_{RMS}	Supplies ±5%
A_{OR}	Output Dynamic Range, VF_R	2.14	2.17	2.20	V_{RMS}	23°C, Nominal Supplies
A_{ORT}	A_{OR} Variation with Temperature			−0.5	mV_{RMS}/°C	Relative to 23°C
A_{ORS}	A_{OR} Variation with Supplies			±18	mV_{RMS}	Supplies ±5%

SUPPLY REJECTION AND CROSSTALK

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ(1)	Max		
$PSRR_1$	V_{DD} Power Supply Rejection Ratio	45			dB	decoder alone(6)
$PSRR_2$	V_{BB} Power Supply Rejection Ratio	35			dB	decoder alone(6)
$PSRR_3$	V_{CC} Power Supply Rejection Ratio	50			dB	decoder alone(6)
$PSRR_4$	V_{DD} Power Supply Rejection Ratio	50			dB	encoder alone(7)
$PSRR_5$	V_{BB} Power Supply Rejection Ratio	45			dB	encoder alone(7)
$PSRR_6$	V_{CC} Power Supply Rejection Ratio	50			dB	encoder alone(7)
CT_R	Crosstalk Isolation, Receive Side	75	80		dB	(Note 8)
CT_T	Crosstalk Isolation, Transmit Side	75	80		dB	(Note 9)
CAPX	Input Sample and Hold Capacitor	1600	2000	2400	pF	

NOTES:
1. Typical values are for $T_A = 25°C$ and nominal power supply values.
2. If the external auto zero is used N_{IC} has a typical value of −76 dBm0.
3. Tested and guaranteed at 23°C, nominal supplies.
4. D_R of Device Under Test (D.U.T.) driven with repetitive digital word sequence specified in CCITT recommendation G.711. Measurement made at VF_R output.
5. With the D.C. method the positive and negative clipping levels are measured and A_{IR} is calculated. With the A.C. method a sinusoidal input signal to VF_X is used where A_{IR} is measured directly.
6. D.U.T. decoder; impose 200 mV_{PP}, 1.02 KHz on appropriate supply; measurement made at decoder output; decoder in idle channel conditions.
7. D.U.T. encoder, impose 200 mV_{PP}, 1.02 KHz on appropriate supply; meaurement made at encoder output; encoder in idle channel conditions.
8. VF_X of D.U.T encoder = 1.02 KHz, 0 dBm0. Decoder under quiet channel conditions; measurements made at decoder output.
9. VF_X = 0 Vrms. Decoder = 1.02 KHz, 0 dBm0. Encoder under quiet channel conditions; measurement made at encoder output.

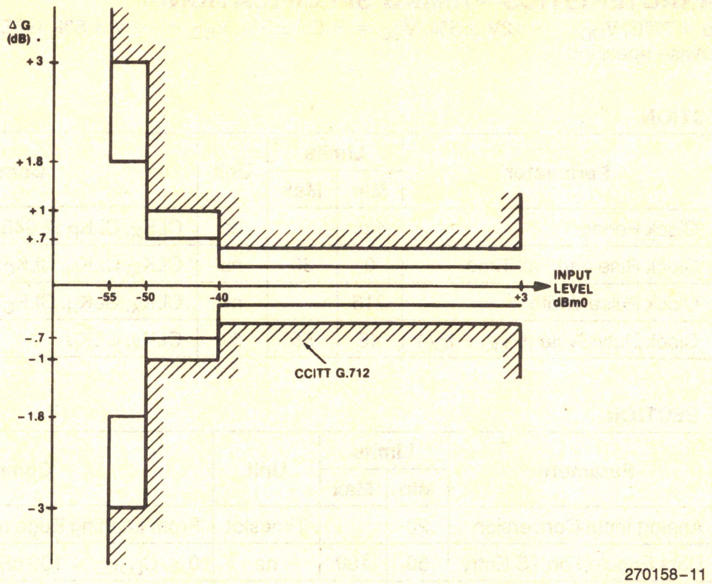


Figure 13. Tracking Deviation (ΔG) (Half Channel)

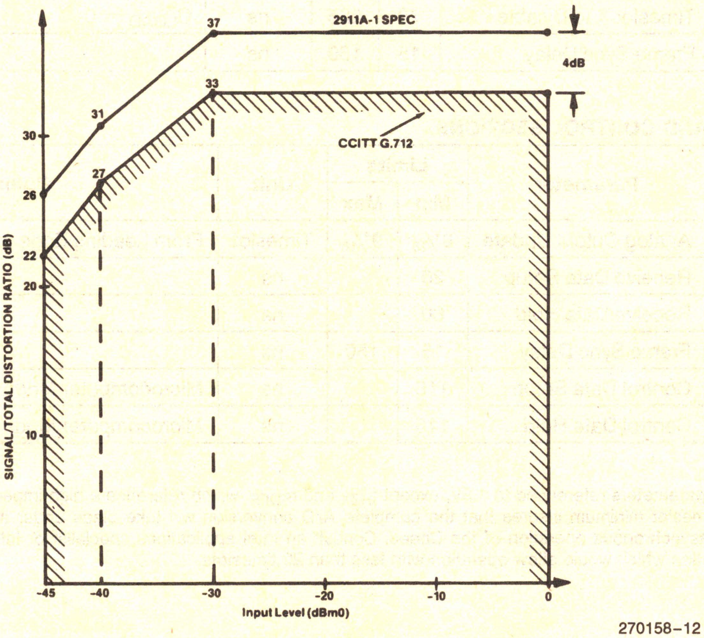


Figure 14. Signal to Total Distortion Ratio (Half Channel)

A.C. CHARACTERISTICS—TIMING SPECIFICATION(1)

T_A = 0°C to +70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified.

CLOCK SECTION

Symbol	Parameter	Limits		Unit	Comments
		Min	Max		
t_{CY}	Clock Period	485		ns	CLK_X, CLK_R (2.048 MHz systems), CLK_C
t_r, t_f	Clock Rise and Fall Time	0	30	ns	CLK_X, CLK_R, CLK_C
t_{CLK}	Clock Pulse Width	215		ns	CLK_X, CLK_R, CLK_C
t_{CDC}	Clock Duty Cycle ($t_{CLK} \div t_{CY}$)	45	55	%	CLK_X, CLK_R

TRANSMIT SECTION

Symbol	Parameter	Limits		Unit	Comments
		Min	Max		
t_{VFX}	Analog Input Conversion	20		Timeslot	From Leading Edge of Transmit Timeslot(2)
t_{DZX}	Data Enabled on TS Entry	50	180	ns	$0 < C_{LOAD} < 100$ pF
t_{DHX}	Data Hold Time	80	230	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	75	245	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	30	185	ns	$0 < C_{LOAD} < 100$ pF
t_{SOFF}	Timeslot X to Disable	70	225	ns	$C_{LOAD} = 0$
t_{FSD}	Frame Sync Delay	15	150	ns	

RECEIVE AND CONTROL SECTIONS

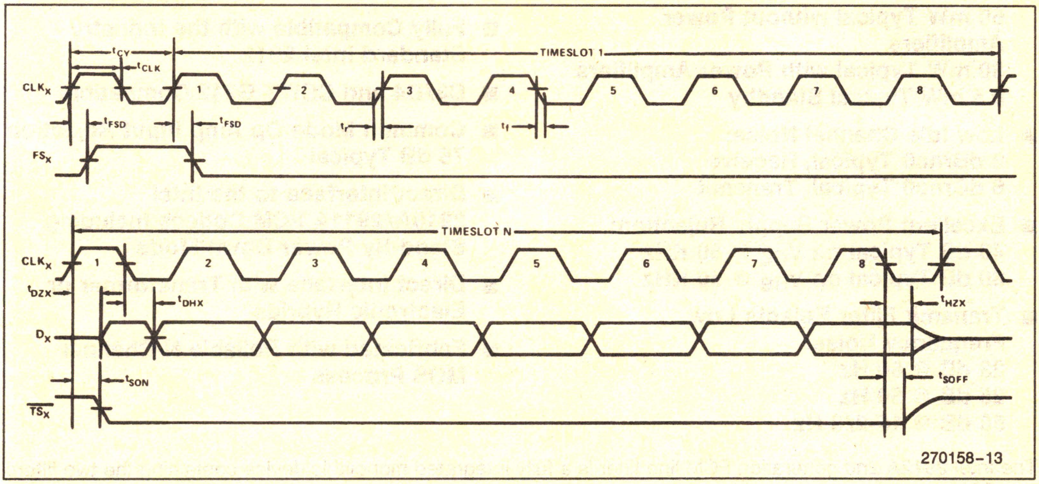
Symbol	Parameter	Limits		Unit	Comments
		Min	Max		
t_{VFR}	Analog Output Update	9 1/16	9 1/16	Timeslot	From Leading Edge of the Channel Timeslot
t_{DSR}	Receive Data Setup	20		ns	
t_{DHR}	Receive Data Hold	60		ns	
t_{FSD}	Frame Sync Delay	15	150	ns	
t_{DSC}	Control Data Setup	115		ns	Microcomputer Mode Only
t_{DHC}	Control Data Hold	115		ns	Microcomputer Mode Only

NOTES:

1. All timing parameters referenced to 1.5V, except t_{HZX} and t_{SOFF}, which reference a high impedance state.
2. The 20 timeslot minimum insures that the complete A/D conversion will take place under any combination of receive interrupt or asynchronous operation of the Codec. Consult an Intel applications specialist or Intel Corporation for applications information which would allow operation with less than 20 timeslots.

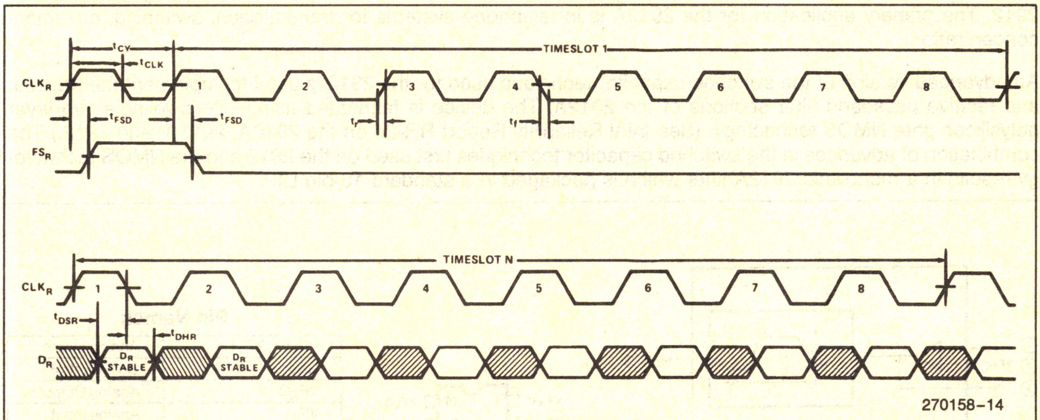
TIMING WAVEFORMS(1)

TRANSMIT TIMING



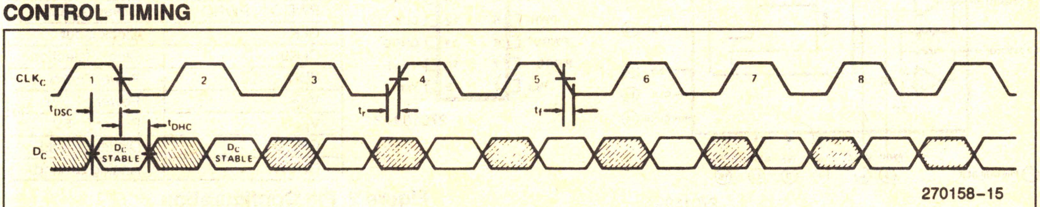
270158-13

RECEIVE TIMING



270158-14

CONTROL TIMING



270158-15

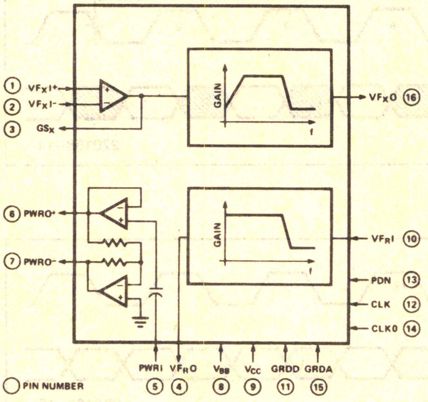
NOTE:
1. All timing parameters referenced to 1.5V, except t_{HZX} and t_{SOFF} which reference a high impedance state.

2912A
PCM TRANSMIT/RECEIVE FILTER

- **Low Power Consumption:**
  60 mW Typical without Power Amplifiers
  80 mW Typical with Power Amplifiers
  0.5 mW Typical Standby
- **Low Idle Channel Noise:**
  2 dBrnc0 Typical, Receive
  6 dBrnc0 Typical, Transmit
- **Excellent Power Supply Rejection:**
  40 dB Typical on V_{CC} @ 50 KHz
  30 dB Typical on V_{BB} @ 50 KHz
- **Transmit Filter Rejects Low Frequency Noise:**
  23 dB @ 60 Hz
  25 dB @ 50 Hz
  50 dB @ 16-2/3 Hz
- **Adjustable Gain in Both Directions**
- **Fully Compatible with the Industry Standard Intel 2912**
- **D3/D4 and CCITT G712 Compatible**
- **Common Mode Op Amp Input Rejection 75 dB Typical**
- **Direct Interface to the Intel 2910A/2911A PCM Codecs Including Stand-By Power Down Mode**
- **Direct Interface with Transformer or Electronic Hybrids**
- **Fabricated with Reliable N-Channel MOS Process**

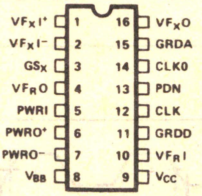
The Intel 2912A 2nd generation PCM line filter is a fully integrated monolithic device containing the two filters of a PCM line or trunk termination. It has improved key parameters of power consumption, idle channel noise, and power supply rejection. A single part exceeds both AT&T* D3/D4 and CCITT transmission specs, exceeds digital Class 5 central office switching system stringent specifications, and is fully compatible with the 2912. The primary application for the 2912A is in telephone systems for transmission, switching, or remote concentration.

An advanced version of the switched capacitor technique used for the 2912 is used to implement the transmit and receive passband filter sections of the 2912A. The device is fabricated using Intel's reliable two layer polysilicon gate NMOS technology. (See Intel Reliability Report RR-24 on the 2910A, 2911A, and 2912.) The combination of advances in the switched capacitor techniques first used on the 2912 and the NMOS technology results in a monolithic 2912A filter which is packaged in a standard 16-pin DIP.



270159–1

Figure 1. Block Diagram



270159–2

Pin Names

VF_XI^+, VF_XI^-	Analog Inputs
GS_X	Gain Control
VF_XO	Analog Output
VF_RI	Analog Input
VF_RO	Analog Output
PWRI	Driver Input
$PWRO^+$, $PWRO^-$	Driver Output
CLK	Clock Input
CLKO	Clock Selection
PDN	Power Down
V_{CC}	Power (+5V)
V_{BB}	Power (−5V)
GRDD	Digital Ground
GRDA	Analog Ground

Figure 2. Pin Configuration

*AT&T is a registered trademark of American Telephone and Telegraph Corporation.

November 1986
Order Number: 270159-001

Table 1. Pin Description

Symbol	Pin No.	Function	Description
VF_XI+	1	Input	Analog input of the transmit filter. The VF_XI+ signal comes from the 2 to 4 wire hybrid in the case of a 2 wire line and goes through the frequency rejection and the antialiasing filters before being sent to the Codec for encoding.
VF_XI-	2	Input	Inverting input of the gain adjustment operational amplifier on the transmit filter.
GS_X	3	Output	Output of the gain adjustment operational amplifier on the transmit filter. Used for gain setting of the transmit filter.
VF_RO	4	Output	Analog output of the receive filter. This output provides a direct interface to electronic hybrids. For a transformer hybrid application, VF_RO is tied to PRWI and a dual balanced output is provided on pins PWRO+ and PWRO−.
PWRI	5	Input	Input to the power driver amplifiers on the receive side for interface to transformer hybrids. High impedance input. When tied to V_{BB}, the power amplifiers are powered down.
PWRO+	6	Output	Non-inverting side of the power amplifiers. Power driver output capable of directly driving transformer hybrids.
PWRO−	7	Output	Inverting side of the power amplifiers. Power driver output capable of directly driving transformer hybrids.
V_{BB}	8	Power	−5V ±5% referenced to GRDA
V_{CC}	9	Power	+5V ±5% referenced to GRDA
VF_RI	10	Input	Analog input of the receive filter, interface to the Codec analog output for PCM applications. The receive filter provides the $\frac{\text{Sinx}}{\text{x}}$ correction needed for sample and hold type Codec outputs to give unity gain. The input voltage range is directly compatible with the Intel 2910A and 2911A Codecs.
GRDD	11	Ground	Digital ground return for internal clock generator.
CLK(1)	12	Input	Clock input. Three clock frequencies can be used: 1.536 MHz, 1.544 MHz or 2.048 MHz; pin 14, CLK0, has to be strapped accordingly. High impedance input, TTL voltage levels.
PDN	13	Input	Control input for the stand-by power down mode. An internal pull up to +5V is provided for interface to the Intel 2910A and 2911A PDN outputs. TTL voltage levels.
CLK0(1)	14	Input	Clock (pin 12, CLK) frequency selection. If tied to V_{BB}, CLK should be 1.536 MHz. If tied to Ground, CLK should be 1.544 MHz. If tied to V_{CC}, CLK should be 2.048 MHz.
GRDA	15	Ground	Analog return common to the transmit and receive analog circuits. Not connected to GRDD internally.
VF_XO	16	Output	Analog output of the transmit filter. The output voltage range is directly compatible with the Intel 2910A and 2911A Codecs.

NOTE:
1. The three clock frequencies are directly compatible with the Intel 2910A and 2911A Codecs. The following table should be observed in selecting the clock frequency.

Codec Clock	Clock Bits/Frame	CLK, Pin 12	CLK0, Pin 14
1.536 MHz	192	1.536 MHz	V_{BB} (−5V)
1.544 MHz	193	1.544 MHz	GRDD
2.048 MHz	256	2.048 MHz	V_{CC} (+5V)

FUNCTIONAL DESCRIPTION

The 2912A provides the transmit and receive filters found on the analog termination of a PCM line or trunk. The transmit filter performs the anti-aliasing function needed for an 8 KHz sampling system, and the 50/60 Hz rejection. The receive filter has a low pass transfer characteristic and also provides the Sinx/x correction necessary to interface the Intel 2910A (μ Law) and 2911A (A Law) Codecs which have a non-return-to-zero output of the digital to analog conversion. Gain adjustment is provided in the receive and transmit directions.

A stand-by, power down mode is included in the 2912A and can be directly controlled by the 2910A/2911A Codecs.

The 2912A can interface directly with a transformer hybrid (2 to 4 wire conversion) or with electronic hybrids; in the latter case the power dissipation is reduced by powering down the output amplifier provided on the 2912A.

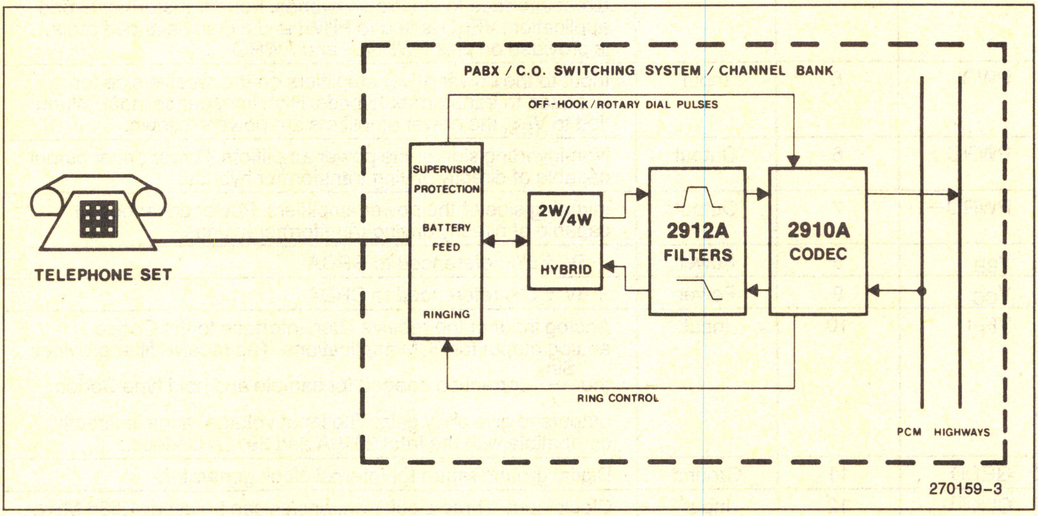


Figure 3. Typical Line Termination

FILTER OPERATION

Transmit Filter Input Stage

The input stage provides gain adjustment in the pass-band. The input operational amplifier has a common mode range of ±2.2 volts, a DC offset of less than 25 mV, a voltage gain greater than 3000 and a unity gain bandwidth of 1 MHz. It can be connected to provide a gain of 20 dB without degrading the noise performance of the filter. The load impedance connected to the amplifier output (GS_X) must be greater than 10K Ω in parallel with 25 pF. The input signal on lead VF_XI + can be either AC or DC coupled. The input Op Amp can also be used in the inverting mode or differential amplifier mode. The remaining portion of the transmit filter provides a gain of +3 dB in the pass band.

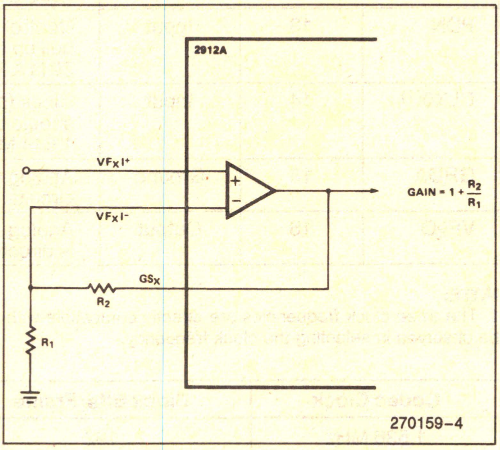


Figure 4. Transmit Filter Gain Adjustment

Receive Filter Output

The VF_RO lead is capable of driving high impedance electronic hybrids. The gain of the receive section from VF_RI to VF_RO is:

$$\frac{\left(\frac{\pi f}{8000}\right)}{\sin\left(\frac{\pi f}{8000}\right)}$$

which when multiplied by the output response of the Intel 2910A and 2911A Codecs results in a 0 dB gain in the pass band. The filter gain can be adjusted downward by a resistor voltage divider connected as shown in Figure 5. The total resistive load R_{LR} on VF_RO should not be less than 10K Ω.

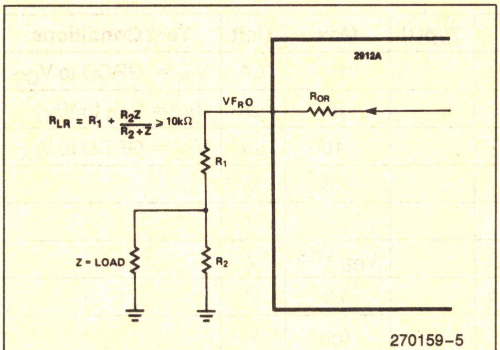


Figure 5. Receive Filter Output Gain Adjustment

Receive Filter Output Driver Amplifier Stage

A balanced power amplifier is provided in order to drive low-impedance loads in a bridged configuration. The receive filter output VF_RO is connected through gain setting resistors R_1 and R_2 to the amplifier input PWRI. The input voltage range on PWRI is ±3.2 volts and the gain is 6 dB for a bridged output.

With a 600Ω load connected between PWRO+ and PWRO−, the maximum voltage swing across the load is ±5.0 volts. The series combination of R_S and the hybrid transformer must present a minimum A.C. load resistance of 600Ω to the amplifier in the bridged configuration. A typical connection of the output driver amplifiers is shown in Figure 6. These amplifiers can also be used with loads connected to ground.

When the power amplifier is not needed it should be deactivated to save power. This is accomplished by tying the PWRI pin to V_{BB} before the device is powered up.

Power Down Mode

Pin 13, PDN, provides the power down control. When the signal on this lead is brought high, the 2912A goes into a standby, power down mode. Power dissipation is reduced to 0.5 mW. In the stand-by mode, all outputs go into a high impedance state. This feature allows multiple 2912As to drive the same analog bus on a time-shared basis.

When power is restored, the settling time of the 2912A is typically 15 ms.

The PDN interface is directly compatible with the Intel 2910A and 2911A PDN outputs. Only one command from the common control is then necessary to power down both the Codec and the Filters of the line or trunk interface.

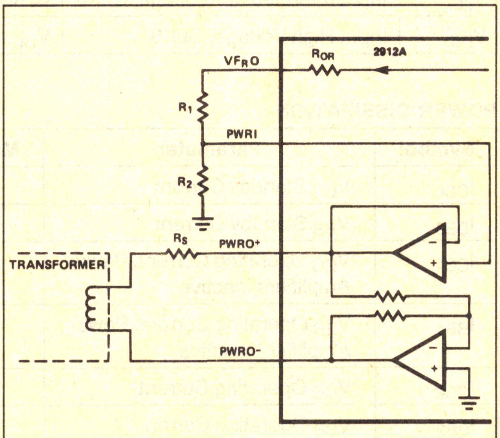


Figure 6. Typical Connection of Output Driver Amplifier

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
Supply Voltage with Respect to V_{BB} .... −0.3V to +14.0V
All Input and Output Voltages with Respect to V_{BB} .... −0.3V to +14.0V
All Output Currents .... ±50 mA
Power Dissipation .... 1 Watt

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to +70°C; V_{CC} = 5V ±5%; V_{BB} = −5V ±5%; GRDA = 0V; GRDD = 0V; unless otherwise specified

DIGITAL INTERFACE (CLK, CLK0, and PDN Pins)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
I_{LIC}	Input Load Current, CLK			10	μA	V_{IN} = GRDD to V_{CC}
I_{LIO}	Input Load Current, CLK0			10	μA	V_{IN} = V_{BB} to V_{CC}
I_{LIP}	Input Load Current, PDN			−100	μA	V_{IN} = GRDD to V_{CC}
V_{IL}	Input Low Voltage (except CLK0)			0.8	V	
V_{IH}	Input High Voltage (except CLK0)	2.0			V	
V_{ILO}	Input Low Voltage, CLK0	V_{BB}		$V_{BB}+0.5$	V	
V_{IIO}	Input Intermediate Voltage, CLK0	GRDD−0.5		0.8	V	
V_{IHO}	Input High Voltage, CLK0	$V_{CC}-0.5$		V_{CC}	V	

POWER DISSIPATION

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
I_{CC0}	V_{CC} Standby Current		50	100	μA	PDN = V_{IH} Min
I_{BB0}	V_{BB} Standby Current		50	100	μA	PDN = V_{IH} Min
I_{CC1}	V_{CC} Operating Current, Power Amplifiers Inactive		6	10	mA	PWRI = V_{BB}(2)
I_{BB1}	V_{BB} Operating Current, Power Amplifiers Inactive		6	10	mA	PWRI = V_{BB}(2)
I_{CC2}	V_{CC} Operating Current		8	14	mA	
I_{BB2}	V_{BB} Operating Current		8	14	mA	

NOTES:
1. Typical values are for T_A = 25°C and nominal power supply values.
2. To place the power amplifiers in the inactive mode PWRI must be tied to V_{BB} prior to power-up.

D.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$; $V_{CC} = 5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0V; GRDD = 0V; unless otherwise specified (Continued)

ANALOG INTERFACE, TRANSMIT FILTER INPUT STAGE

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
I_{BXI}	Input Leakage Current, VF_XI+, VF_XI-			100	nA	$-2.2V < V_{IN} < 2.2V$
R_{IXI}	Input Resistance, VF_XI+, VF_XI-	10			MΩ	
V_{OSXI}	Input Offset Voltage, VF_XI+, VF_XI-			25	mV	
CMRR	Common Mode Rejection, VF_XI+, VF_XI-	60	75		dB	$-2.2V < V_{IN} < 2.2V$, 0 dBm0 ≡ 1.1 V_{RMS}, Input at VF_XI-
A_{VOL}	DC Open Loop Voltage Gain, GS_X	3000				
f_C	Open Loop Unity Gain Bandwidth, GS_X		1		MHz	
V_{OXI}	Output Voltage Swing, GS_X	±2.5			V	$R_L \geq 10$ KΩ
C_{LXI}	Load Capacitance, GS_X			25	pF	
R_{LXI}	Minimum Load Resistance, GS_X	10			KΩ	Minimum R_L

ANALOG INTERFACE, TRANSMIT FILTER (See Figure 9)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
R_{OX}	Output Resistance, VF_XO		20	35	Ω	
V_{OSX}	Output DC Offset, VF_XO			100	mV	VF_XI+ Connected to GRDA, Input Op Amp at Unity Gain
$PSRR_1$	Power Supply Rejection of V_{CC} at 1 KHz, VF_XO	30	40		dB	Note 2
$PSRR_2$	Power Supply Rejection of V_{BB} at 1 KHz, VF_XO	25	30		dB	Note 2
C_{LX}	Load Capacitance, VF_XO			25	pF	
R_{LX}	Minimum Load Resistance, VF_XO	2.7			KΩ	Minimum R_L
V_{OX1}	Output Voltage Swing, 1 KHz, VF_XO	±3.2			V	$R_L \geq 10$ KΩ or with 2910A or 2911A
V_{OX2}	Output Voltage Swing, 1 KHz, VF_XO	±2.5			V	$R_L \geq 2.7$ KΩ

NOTES:
1. Typical values for $T_A = 25°C$ and nominal power supply values.
2. $PSRR_{1,2}$ include op amp in transmit section.

D.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$; $V_{CC} = 5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0V; GRDD = 0V; unless otherwise specified (Continued)

ANALOG INTERFACE, RECEIVE FILTER (See Figure 10)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
I_{BR}	Input Leakage Current, VF_RI			3	μA	$-3.2V < V_{IN} < 3.2V$
R_{IR}	Input Resistance, VF_RI	1			MΩ	
R_{OR}	Output Resistance, VF_RO			100	Ω	
V_{OSR}	Output DC Offset VF_RO			100	mV	VF_RI Connected to GRDA
$PSRR_3$	Power Supply Rejection of V_{CC} at 1 KHz, VF_RO	30	45		dB	
$PSRR_4$	Power Supply Rejection of V_{BB} at 1 KHz, VF_RO	30	35		dB	
C_{LR}	Load Capacitance, VF_RO			25	pF	
R_{LR}	Minimum Load Resistance, VF_RO	10			KΩ	Minimum R_L
V_{OR}	Output Voltage Swing, VF_RO	±3.2			V	$R_L = 10\ K\Omega$

ANALOG INTERFACE, RECEIVE FILTER DRIVER AMPLIFIER STAGE

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions	
I_{BRA}	Input Leakage Current, PWRI			3	μA	$-3.2V < V_{IN} < 3.2V$	
R_{IRA}	Input Resistance, PWRI	10			MΩ		
R_{ORA}	Output Resistance, PWRO+, PWRO−		1		Ω	$\lvert I_{OUT}\rvert < 10$ mA $-3.0V < V_{OUT} < 3.0V$	
V_{OSRA}	Output DC Offset, PWRO+, PWRO−			50	mV	PWRI Connected to GRDA	
C_{LRA}	Load Capacitance, PWRO+, PWRO−			100	pF		
V_{ORA1}	Output Voltage Swing Across R_L, PWRO+, PWRO− Single Ended Connection	±3.2			V	$R_L = 10\ K\Omega$	R_L Connected to GRDA fo ≥ 200 Hz
		±2.9			V	$R_L = 600\Omega$	
		±2.5			V	$R_L = 300\Omega$	
V_{ORA2}	Differential Output Voltage Swing, PWRO+, PWRO− Balanced Output Connection	±6.4			V	$R_L = 20\ K\Omega$	R_L Connected between PWRO + and PWRO − fo ≥ 200 Hz
		±5.8			V	$R_L = 1200\Omega$	
		±5.0			V	$R_L = 600\Omega$	

NOTE:
1. Typical values are for $T_A = 25°C$ and nominal power supply values.

A.C. CHARACTERISTICS T_A = 0°C to +70°C; V_{CC} = 5V ±5%; V_{BB} = −5V ±5%; GRDA = 0V; GRDD = 0V; unless otherwise specified

Clock Input Frequency: CLK = 1.536 MHz ±0.1%; CLK0 = V_{IL0} (Tied to V_{BB})
CLK = 2.048 MHz ±0.1%; CLK0 = V_{IH0} (Tied to V_{CC})
CLK = 1.544 MHz ±0.1%; CLK0 = V_{II0} (Tied to GRDD)

TRANSMIT FILTER TRANSFER CHARACTERISTICS
(See Transmit Filter Transmit Characteristics, Figure 7)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
G_{RX}	Gain Relative to Gain at 1 KHz					0 dBm0 Input Signal
	16.67 Hz		−56	−50	dB	Gain Setting Op Amp
	50 Hz			−25	dB	Unity Gain
	60 Hz			−23	dB	
	200 Hz	−1.0		−0.125	dB	0 dBm0 Signal ≡ 1.1 V_{RMS}
	300 Hz to 3000 Hz	−0.125		0.125	dB	Input at VF_XI-
	3300 Hz	−0.35		0.03	dB	
	3400 Hz	−0.7		−0.1	dB	0 dBm0 Signal ≡ 1.6 V_{RMS}
	4000 Hz			−14	dB	Output at VF_XO
	4600 Hz and Above			−32	dB	
G_{AX}	Absolute Passband Gain at 1 KHz, VF_XO	2.9	3.0	3.1	dB	$R_L = \infty$(3)
G_{AXT}	Gain Variation with Temperature at 1 KHz		0.0002	0.002	dB/°C	0 dBm0 Signal Level
G_{AXS}	Gain Variation with Supplies at 1 KHz		0.01	0.07	dB/V	0 dBm0 Signal Level, Supplies ±5%
CT_{RT}	Cross Talk, Receive to Transmit, Measured at VF_XO $20 \log \frac{VF_XO}{VF_RO}$		−75	−65	dB	VF_RI = 1.6 V_{RMS}, 1 KHz Input, VF_XI+, VF_XI- Connected to GS_X, GS_X Connected through 10 KΩ to GRDA
N_{CX1}	Total C Message Noise at Output, VF_XO		6	11	dBrnc0 (Note 2)	Gain Setting Op Amp at Unity Gain
N_{CX2}	Total C Message Noise at Output, VF_XO		9	13	dBrnc0 (Note 2)	Gain Setting Op Amp at 20 dB Gain
D_{DX}	Differential Envelope Delay, VF_XO 1 KHz to 2.6 KHz			60	μs	
D_{AX}	Absolute Delay at 1 KHz, VF_XO			110	μs	
DP_{X1}	Single Frequency Distortion Products			−48	dB	0 dBm0 Input Signal at 1 KHz
DP_{X2}	Single Frequency Distortion Products at Maximum Signal Level of +3 dBm0 at VF_XO			−45	dB	0.16 V_{RMS} 1 KHz Input Signal at VF_XI+; Gain Setting Op Amp at 20 dB Gain. The +3 dBm0 Signal at VF_XO is 2.26 V_{RMS}

A.C. CHARACTERISTICS $T_A = 0°C$ to $+70°C$; $V_{CC} = 5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0V; GRDD = 0V; unless otherwise specified (Continued)

Clock Input Frequency: CLK = 1.536 MHz ±0.1%; CLK0 = V_{IL0} (Tied to V_{BB})
CLK = 1.544 MHz ±0.1%; CLK0 = V_{II0} (Tied to GRDD)
CLK = 2.048 MHz ±0.1%; CLK0 = V_{IH0} (Tied to V_{CC})

RECEIVE FILTER TRANSFER CHARACTERISTICS (See Receive Filter Transfer Characteristics, Figure 8)

Symbol	Parameter	Min	Typ(1)	Max	Unit	Test Conditions
G_{RR}	Gain Relative to Gain at 1 KHz with Sinx/x Correction of 2910A or 2911A					0 dBm0 Input Signal
	Below 200 Hz			0.125	dB	0 dBm0 Signal ≡ 1.6 V_{RMS} × $\text{Sin}\left(\frac{\pi f}{8000}\right)$
	200 Hz	−0.5		0.125	dB	
	300 Hz to 3000 Hz	−0.125		0.125	dB	
	3300 Hz	−0.35		0.03	dB	Input at VF_RI $\left(\frac{\pi f}{8000}\right)$
	3400 Hz	−0.7		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−30	dB	
G_{AR}	Absolute Passband Gain at 1 KHz, VF_RO	−0.1	0	+0.1	dB	$R_L = \infty$ (3, 4)
G_{ART}	Gain Variation with Temperature at 1 KHz		0.0002	0.002	dB/°C	0 dBm0 Signal Level
G_{ARS}	Gain Variation with Supplies at 1 KHz		0.01	0.07	dB/V	0 dBm0 Signal Level, Supplies ±5%
CT_{TR}	Cross Talk, Transmit to Receive, Measured at VF_RO; 20 log (VF_RO/VF_XO)		−70	−60	dB	VF_XI = 1.1 V_{RMS}, 1 KHz Output, VF_RI Connected to GRDA
N_{CR}	Total C Message Noise at Output, VF_RO		2	6	dBrnc0 (Note 2)	VF_RO Output or PWRO+ and PWRO− Connected with Unity Gain
D_{DR}	Differential Envelope Delay, VF_RO, 1 KHz to 2.6 KHz			100	μs	
D_{AR}	Absolute Delay at 1 KHz, VF_RO			110	μs	
DP_{R1}	Single Frequency Distortion Products			−48	dB	0 dBm0 Input Signal at 1 KHz
DP_{R2}	Single Frequency Distortion Products at Maximum Signal Level of +3 dBm0 at VF_RO			−45	dB	+3 dBm0 Signal Level of 2.26 V_{RMS}, 1 KHz Input at VF_RI

NOTES:
1. Typical Values are for T_A = 25°C and nominal power supply values.
2. A noise measurement of 12 dBrnc into a 600Ω load at the 2912A device is equivalent to 6 dBrnc0.
3. For gain under load refer to output resistance specs and perform gain calculation.
4. Output is non-inverting.

TRANSFER CHARACTERISTICS

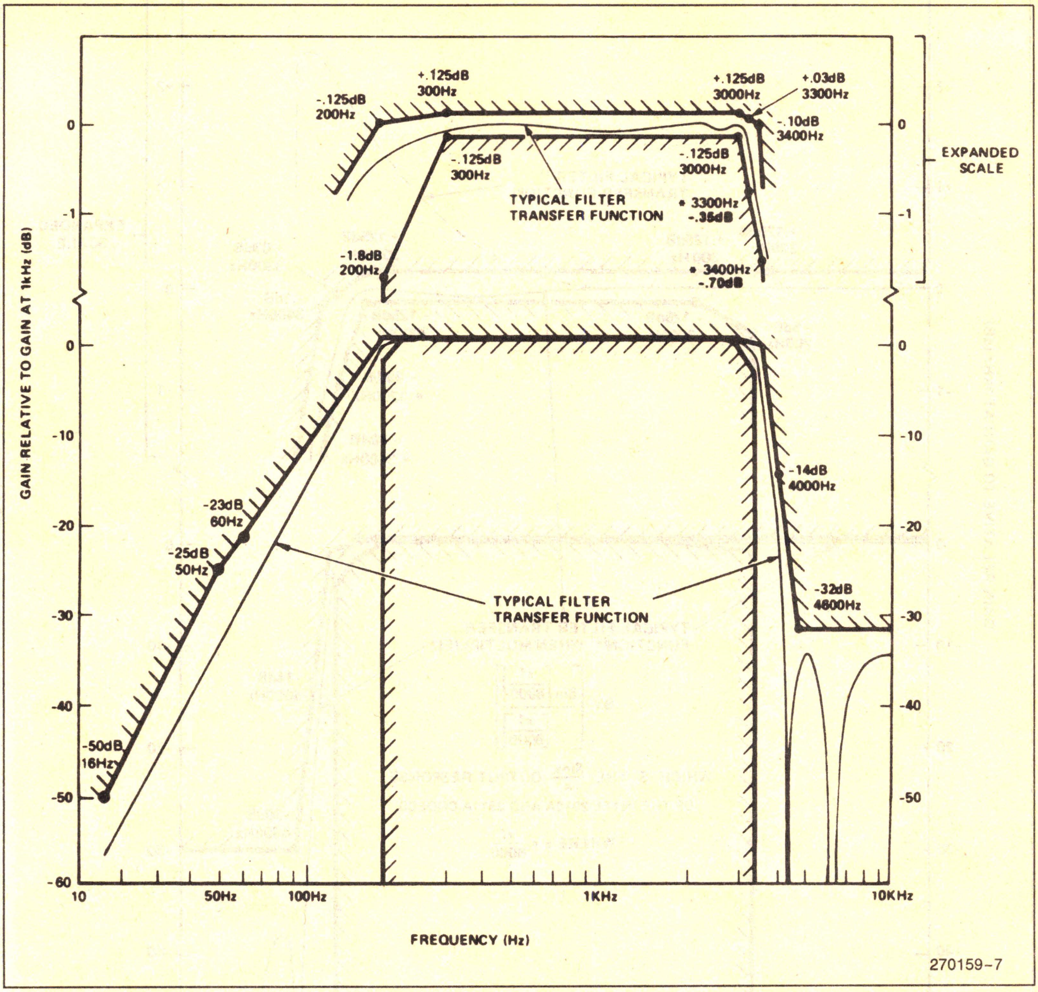


Figure 7. Transmit Filter

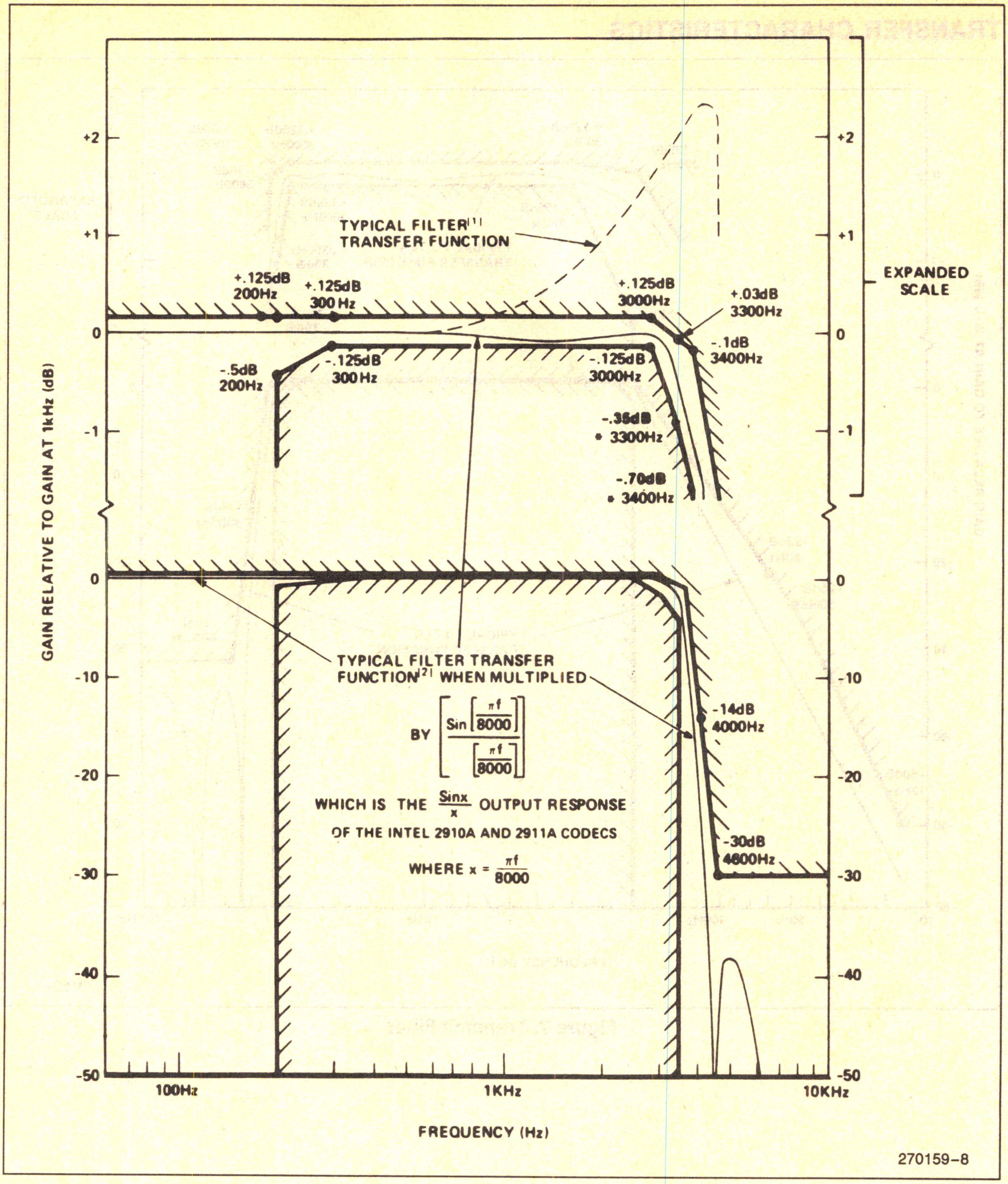


Figure 8. Receive Filter

NOTES:

1. Typical Transfer Function of the Receive Filter as a Separate Component.
2. Typical Transfer Function of the Receive Filter Driven by the Sample and Hold Output of the Intel 2910A and 2911A CODECS. The Combined Filter/CODEC Response Meets the Stated Specifications.

POWER SUPPLY REJECTION TYPICAL VALUES OVER 3 RANGES

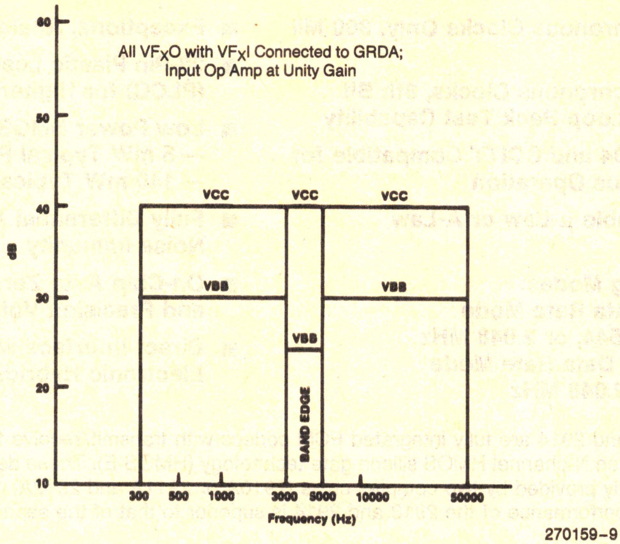


Figure 9. Transmit Filter

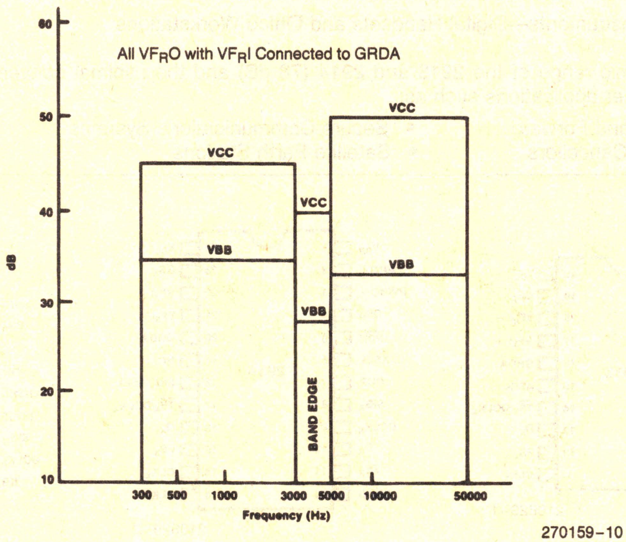


Figure 10. Receive Filter

2913 AND 2914
COMBINED SINGLE-CHIP PCM CODEC AND FILTER

- 2913 Synchronous Clocks Only, 300 Mil Package
- 2914 Asynchronous Clocks, 8th Bit Signaling, Loop Back Test Capability
- AT&T D3/D4 and CCITT Compatible for Synchronous Operation
- Pin Selectable μ-Law or A-Law Operation
- Two Timing Modes:
  — Fixed Data Rate Mode 1.536, 1.544, or 2.048 MHz
  — Variable Data Rate Mode 64 KHz 2.048 MHz
- Exceptional Analog Performance
- 28-Pin Plastic Leaded Chip Carrier (PLCC) for Higher Integration
- Low Power HMOS-E Technology:
  — 5 mW Typical Power Down
  — 140 mW Typical Operating
- Fully Differential Architecture Enhances Noise Immunity
- On-Chip Auto Zero, Sample and Hold, and Precision Voltage References
- Direct Interface with Transformer or Electronic Hybrids

The Intel 2913 and 2914 are fully integrated PCM codecs with transmit/receive filters fabricated in a highly reliable and proven N-channel HMOS silicon gate technology (HMOS-E). These devices provide the functions that were formerly provided by two complex chips (2910A or 2911A and 2912A). Besides the higher level of integration, the performance of the 2913 and 2914 is superior to that of the separate devices.

The primary applications for the 2913 and 2914 are in telephone systems:

- Switching—Digital PBX's and Central Office Switching Systems
- Transmission—D3/D4 Type Channel Banks and Subscriber Carrier Systems
- Subscriber Instruments—Digital Handsets and Office Workstations

The wide dynamic range of the 2913 and 2914 (78 dB) and the minimal conversion time make them ideal products for other applications such as:

- Voice Store and Forward
- Digital Echo Cancellers
- Secure Communications Systems
- Satellite Earth Stations

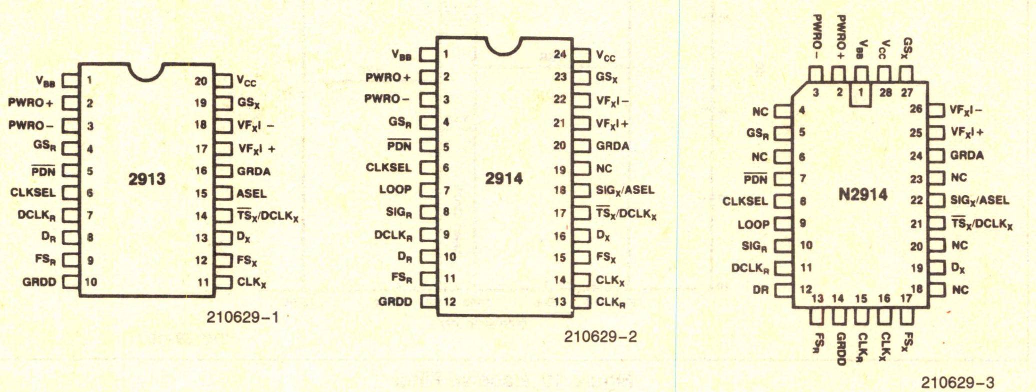


Figure 1. Pin Configurations

November 1986
Order Number: 210629-002

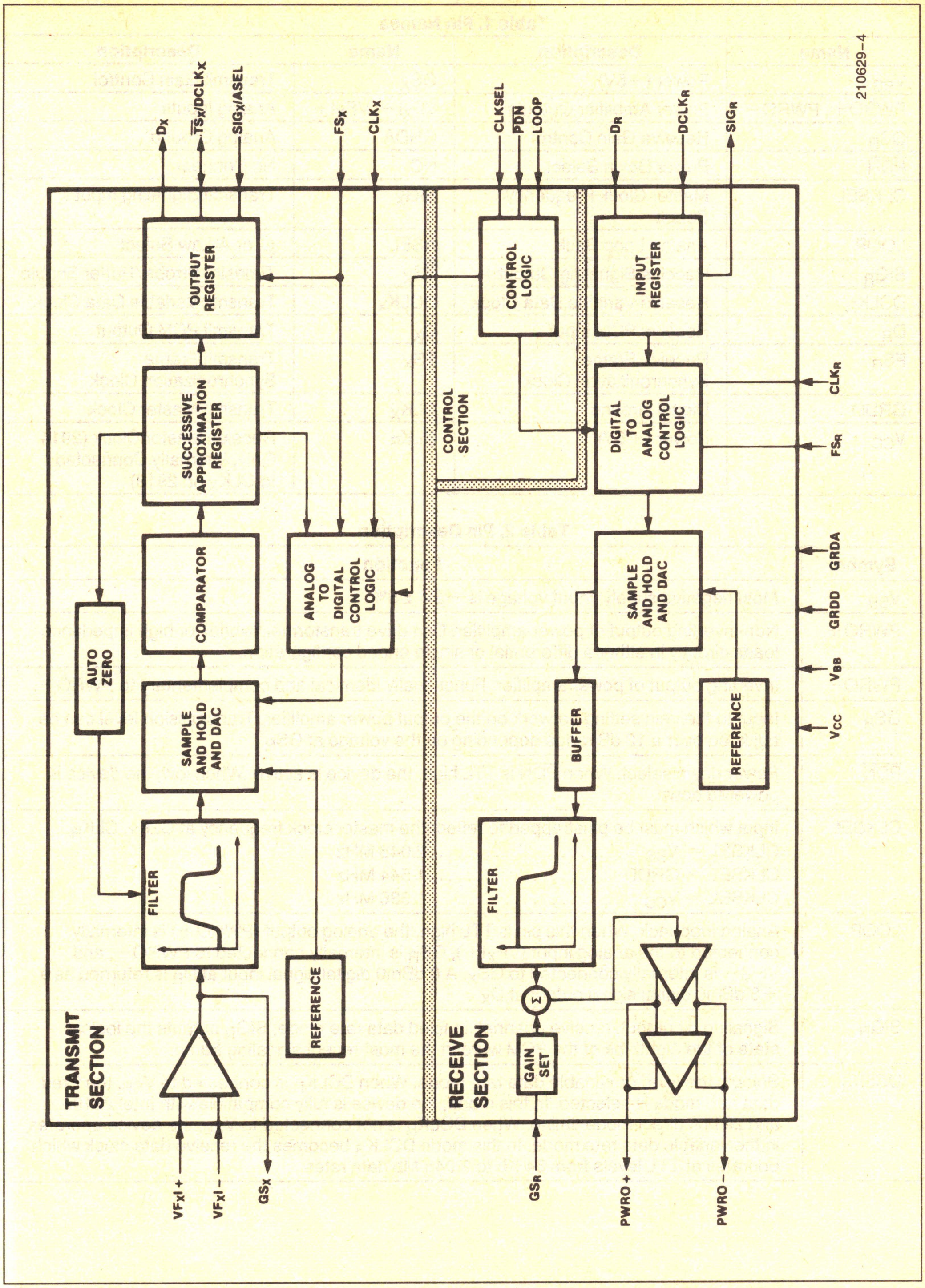


Figure 2. Block Diagram

Table 1. Pin Names

Name	Description	Name	Description
V_{BB}	Power (−5V)	GS_X	Transmit Gain Control
PWRO+, PWRO−	Power Amplifier Outputs	VF_XI-, VF_XI+	Analog Inputs
GS_R	Receive Gain Control	GRDA	Analog Ground
$\overline{PDN}$	Power Down Select	NC	No Connect
CLKSEL	Master Clock Frequency Select	SIG_X	Transmit Signaling Input
LOOP	Analog Loop Back	ASEL	μ- or A-Law Select
SIG_R	Receive Signaling Output	$\overline{TS}_X$	Timeslot Strobe/Buffer Enable
$DCLK_R$	Receive Variable Data Clock	$DCLK_X$	Transmit Variable Data Clock
D_R	Receive PCM Input	D_X	Transmit PCM Output
FS_R	Receive Frame Synchronization Clock	FS_X	Transmit Frame Synchronization Clock
GRDD	Digital Ground	CLK_X	Transmit Master Clock
V_{CC}	Power (+5V)	CLK_R	Receive Master Clock (2914 Only, Internally Connected to CLK_X on 2913)

Table 2. Pin Description

Symbol	Function
V_{BB}	Most negative supply; input voltage is −5V ±5%.
PWRO+	Non-inverting output of power amplifier. Can drive transformer hybrids or high impedance loads directly in either a differential or single ended configuration.
PWRO−	Inverting output of power amplifier. Functionally identical and complementary to PWRO+.
GS_R	Input to the gain setting network on the output power amplifier. Transmission level can be adjusted over a 12 dB range depending on the voltage at GS_R.
$\overline{PDN}$	Power down select. When $\overline{PDN}$ is TTL high, the device is active. When low, the device is powered down.
CLKSEL	Input which must be pinstrapped to reflect the master clock frequency at CLK_X, CLK_R. CLKSEL = V_{BB} ....2.048 MHz CLKSEL = GRDD ....1.544 MHz CLKSEL = V_{CC} ....1.536 MHz
LOOP	Analog loopback. When this pin is TTL high, the analog output (PWRO+) is internally connected to the analog input (VF_XI+), GS_R is internally connected to PWRO−, and VF_XI- is internally connected to GS_X. A 0 dBm0 digital signal input at D_R is returned as a +3 dBm0 digital signal output at D_X.
SIG_R	Signaling bit output, receive channel. In fixed data rate mode, SIG_R outputs the logical state of the eighth bit of the PCM word in the most recent signaling frame.
$DCLK_R$	Selects the fixed or variable data rate mode. When $DCLK_R$ is connected to V_{BB}, the fixed data rate mode is selected. In this mode, the device is fully compatible with Intel 2910A and 2911A direct mode timing. When $DCLK_R$ is not connected to V_{BB}, the device operates in the variable data rate mode. In this mode $DCLK_R$ becomes the receive data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.

Table 2. Pin Description (Continued)

Symbol	Function
D_R	Receive PCM input. PCM data is clocked in on this lead on eight consecutive negative transitions of the receive data clock; CLK_R in the fixed data rate mode and $DCLK_R$ in variable data rate mode.
FS_R	8 KHz frame synchronization clock input/timeslot enable, receive channel. A multi-function input which in fixed data rate mode distinguishes between signaling and non-signaling frames by means of a double or single wide pulse respectively. In variable data rate mode this signal must remain high for the entire length of the timeslot. The receive channel enters the standby state whenever FS_R is TTL low for 300 milliseconds.
GRDD	Digital ground for all internal logic circuits. Not internally tied to GRDA.
CLK_R	Receive master and data clock for the fixed data rate mode; receive master clock only in variable data rate mode.
CLK_X	Transmit master and data clock for the fixed data rate mode; transmit master clock only in variable data rate mode.
FS_X	8 KHz frame synchronization clock input/timeslot enable, transmit channel. Operates independently but in an analogous manner to FS_R. The transmit channel enters the standby state whenever FS_X is TTL low for 300 milliseconds.
D_X	Transmit PCM output. PCM data is clocked out on this lead on eight consecutive positive transitions of the transmit data clock: CLK_X in fixed data rate mode and $DCLK_X$ in variable data rate mode.
$\overline{TS}_X/DCLK_X$	Transmit channel timeslot strobe (output) or data clock (input) for the transmit channel. In fixed data rate mode, this pin is an open drain output designed to be used as an enable signal for a three-state buffer as in 2910A and 2911A direct mode timing. In variable data rate mode, this pin becomes the transmit data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
SIG_X/ASEL	A dual purpose pin. When connected to V_{BB}, A-law operation is selected. When it is not connected to V_{BB} this pin is a TTL level input for signaling operation. This input is transmitted as the eighth bit of the PCM word during signaling frames on the D_X lead. If not used as an input pin, ASEL should be strapped to either V_{CC} or GRDD.
NC	No connect.
GRDA	Analog ground return for all internal voice circuits. Not internally connected to GRDD.
VF_XI+	Non-inverting analog input to uncommitted transmit operational amplifier.
VF_XI-	Inverting analog input to uncommitted transmit operational amplifier.
GS_X	Output terminal of transmit channel input op amp. Internally, this is the voice signal input to the transmit filter.
V_{CC}	Most positive supply; input voltage is $+5V \pm 5\%$.

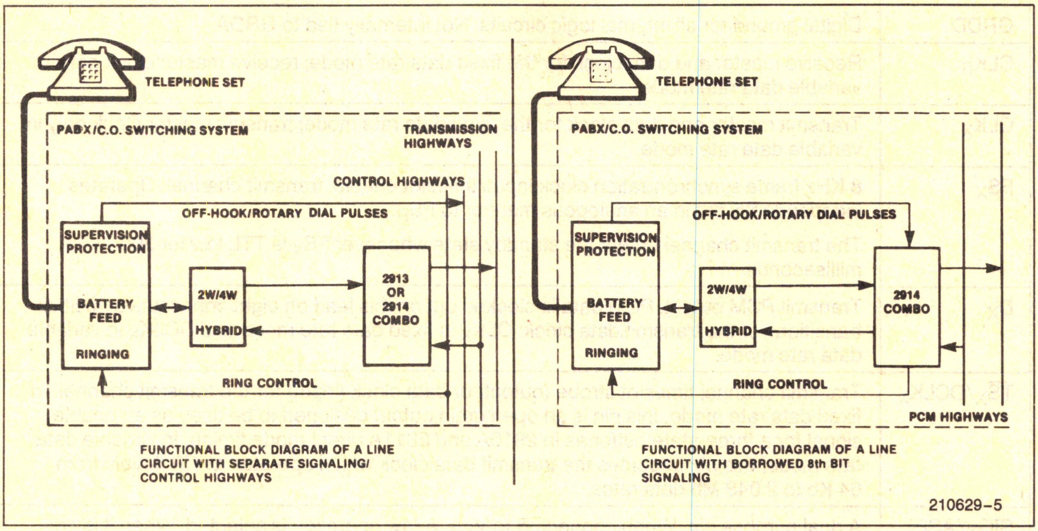
FUNCTIONAL DESCRIPTION

The 2913 and 2914 provide the analog-to-digital and the digital-to-analog conversions and the transmit and receive filtering necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. They are intended to be used at the analog termination of a PCM line or trunk.

The following major functions are provided:

- Bandpass filtering of the analog signals prior to encoding and after decoding
- Encoding and decoding of voice and call progress information
- Encoding and decoding of the signaling and supervision information

SWITCHING



CHANNEL BANKS

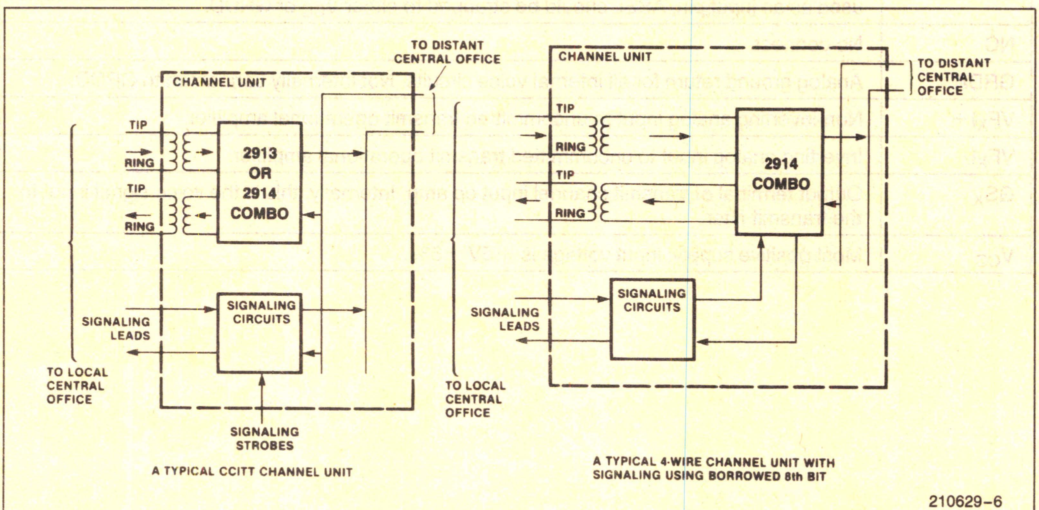


Figure 3. Typical Line Terminations

GENERAL OPERATION

System Reliability Features

The combochip can be powered up by pulsing FS_X and/or FS_R while a TTL high voltage is applied to $\overline{PDN}$, provided that all clocks and supplies are connected. The 2913 and 2914 have internal resets on power up (or when V_{BB} or V_{CC} are re-applied) in order to ensure validity of the digital outputs and thereby maintain integrity of the PCM highway.

On the transmit channel, digital outputs D_X and $\overline{TS}_X$ are held in a high impedance state for approximately four frames (500 μs) after power up or application of V_{BB} or V_{CC}. After this delay, D_X, $\overline{TS}_X$, and signaling will be functional and will occur in the proper timeslot. The analog circuits on the transmit side require approximately 60 milliseconds to reach their equilibrium value due to the autozero circuit settling time. Thus, valid digital information, such as for on/off hook detection, is available almost immediately, while analog information is available after some delay.

On the receive channel, the digital output SIG_R is also held low for a maximum of four frames after power up or application of V_{BB} or V_{CC}. SIG_R will remain low thereafter until it is updated by a signaling frame.

To further enhance system reliability, $\overline{TS}_X$ and D_X will be placed in a high impedance state approximately 30 μs after an interruption of CLK_X. Similarly, SIG_R will be held low approximately 30 μs after an interruption of CLK_R. These interruptions could possibly occur with some kind of fault condition.

Power Down and Standby Modes

To minimize power consumption, two power down modes are provided in which most 2913/2914 functions are disabled. Only the power down, clock, and frame sync buffers, which are required to power up the device, are enabled in these modes. As shown in Table 3, the digital outputs on the appropriate channels are placed in a high impedance state until the device returns to the active mode.

The Power Down mode utilizes an external control signal to the $\overline{PDN}$ pin. In this mode, power consumption is reduced to the value shown in Table 3. The device is active when the signal is high and inactive when it is low. In the absence of any signal, the $\overline{PDN}$ pin floats to TTL high allowing the device to remain active continuously.

The Standby mode leaves the user an option of powering either channel down separately or powering the entire device down by selectively removing FS_X and/or FS_R. With both channels in the standby state, power consumption is reduced to the value shown in Table 3. If transmit only operation is desired, FS_X should be applied to the device while FS_R is held low. Similarly, if receive only operation is desired, FS_R should be applied while FS_X is held low.

Fixed Data Rate Mode

Fixed data rate timing, which is 2910A and 2911A compatible, is selected by connecting $DCLK_R$ to V_{BB}. It employs master clocks CLK_X and CLK_R, frame synchronization clocks FS_X and FS_R, and output $\overline{TS}_X$.

Table 3. Power-Down Methods

Device Status	Power-Down Method	Typical Power Consumption	Digital Output Status
Power Down Mode	$\overline{PDN}$ = TTL Low	5 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state and SIG_R is placed in a TTL low state within 10 μs.
Standby Mode	FS_X and FS_R are TTL Low	12 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state and SIG_R is placed in a TTL low state 300 milliseconds after FS_X and FS_R are removed.
Only Transmit Is on Standby	FS_X is TTL Low	70 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 300 milliseconds.
Only Receive Is on Standby	FS_R is TTL Low	110 mW	SIG_R is placed in a TTL low state within 300 milliseconds.

CLK_X and CLK_R serve both as master clocks to operate the codec and filter sections and bit clocks to clock the data in and out from the PCM highway. FS_X and FS_R are 8 KHz inputs which set the sampling frequency and distinguish between signaling and non-signaling frames by their pulse width. A frame synchronization pulse which is one master clock wide designates a non-signaling frame, while a double wide sync pulse enables the signaling function. $\overline{TS}_X$ is a timeslot strobe/buffer enable output which gates the PCM word onto the PCM highway when an external buffer is used to drive the line.

Data is transmitted on the highway at D_X on the first eight positive transitions of CLK_X following the rising edge of FS_X. Similarly, on the receive side, data is received on the first eight falling edges of CLK_R. The frequency of CLK_X and CLK_R is selected by the CLKSEL pin to be either 1.536, 1.544, or 2.048 MHz. No other frequency of operation is allowed in the fixed data rate mode.

Variable Data Rate Mode

Variable data rate timing is selected by connecting $DCLK_R$ to the bit clock for the receive PCM highway rather than to V_{BB}. It employs master clocks CLK_X and CLK_R, bit clocks $DCLK_R$ and $DCLK_X$, and frame synchronization clocks FS_R and FS_X.

Variable data rate timing allows for a flexible data frequency. It provides the ability to vary the frequency of the bit clocks, which can be asynchronous in the case of the 2914 or synchronous in the case of the 2913, from 64 KHz to 2.048 MHz. Master clock's inputs are still restricted to 1.536, 1.544, or 2.048 MHz.

In this mode, $DCLK_R$ and $DCLK_X$ become the data clocks for the receive and transmit PCM highways. While FS_X is high, PCM data from D_X is transmitted onto the highway on the next eight consecutive positive transitions of $DCLK_X$. Similarly, while FS_R is high, each PCM bit from the highway is received by D_R on the next eight consecutive negative transitions of $DCLK_R$.

On the transmit side, the PCM word will be repeated in all remaining timeslots in the 125 μs frame as long as $DCLK_X$ is pulsed and FS_X is held high. This feature allows the PCM word to be transmitted to the PCM highway more than once per frame, if desired, and is only available in the variable data rate mode. Conversely, signaling is only allowed in the fixed data rate mode since the variable mode provides no means with which to specify a signaling frame.

Signaling

Signaling can only be performed with the 24-pin device in the fixed data rate timing mode ($DCLK_R$ = V_{BB}). Signaling frames on the transmit and receive sides are independent of one another and are selected by a double-width frame sync pulse on the appropriate channel. During a transmit signaling frame, the codec will encode the incoming analog signal and substitute the signal present on SIG_X for the least significant bit of the encoded PCM word. Similarly, in a receive signaling frame, the codec will decode the seven most significant bits according to CCITT recommendation G.733 and output the logical state of the LSB on the SIG_R lead until it is updated in the next signaling frame. Timing relationships for signaling operation are shown in Figure 4.

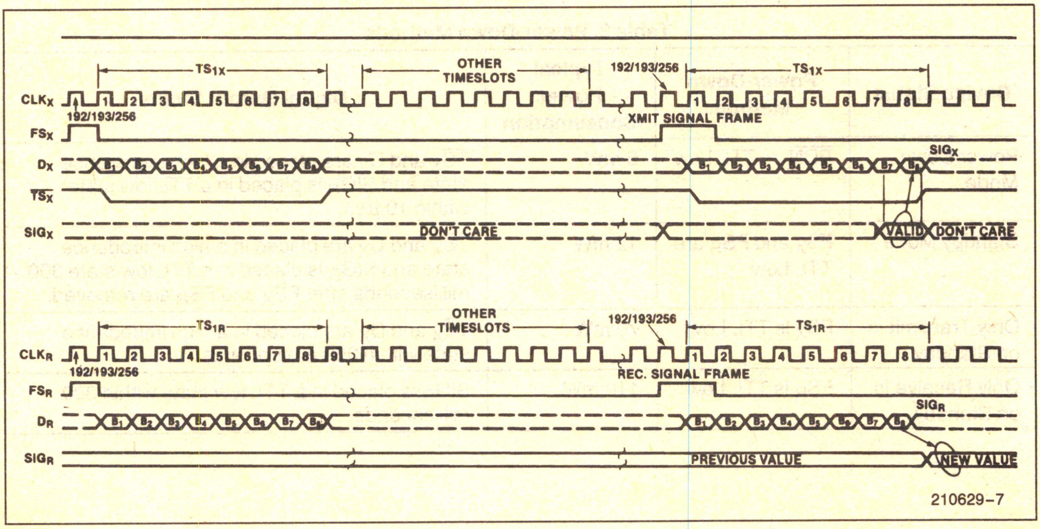


Figure 4. Signaling Timing (Used Only with Fixed Data Rate Mode)

Asynchronous Operation

The 2914 can be operated with asynchronous clocks in either the fixed or variable data rate modes. In order to avoid crosstalk problems associated with special interrupt circuitry, the design of the Intel 2913/2914 combochip includes separate digital-to-analog converters and voltage references on the transmit and receive sides to allow independent operation of the two channels.

In either timing mode, the master clock, data clock, and timeslot strobe must be synchronized at the beginning of each frame. CLK_X and $DCLK_X$ are synchronized once per frame but may be of different frequencies. The receive channel operates in a similar manner and is completely independent of the transmit timing (refer to Variable Data Rate Timing Diagrams). This approach requires the provision of two separate master clocks, even in variable data rate mode, but avoids the use of a synchronizer which can cause intermittent data conversion errors.

Analog Loopback

A distinctive feature of the 2914 is its analog loopback capability. This feature allows the user to send a control signal which internally connects the analog input and output ports. As shown in Figure 5, when LOOP is TTL high the analog output (PWRO+) is internally connected to the analog input (VF_XI+), GS_R is internally connected to PWRO−, and VF_XI- is internally connected to GS_X.

With this feature, the user can test the line circuit remotely by comparing the digital codes sent into the receive channel (D_R) with those generated on the transmit channel (D_X). Due to the difference in transmission levels between the transmit and receive sides, a 0 dBm0 code sent into D_R will emerge from D_X as a +3 dBm0 code, an implicit gain of 3 dB. Thus, the maximum signal input level which can be tested using analog loopback is 0 dBm0.

Precision Voltage References

No external components are required with the combochip to provide the voltage reference function. Voltage references are generated on-chip and are calibrated during the manufacturing process. The technique uses a difference in sub-surface charge density between two suitably implanted MOS devices to derive a temperature and bias stable reference voltage. These references determine the gain and dynamic range characteristics of the device.

Separate references are supplied to the transmit and receive sections and each is trimmed independently during the manufacturing process. The reference value is then further trimmed in the gain setting op-amps to a final precision value. With this method the combochip can achieve the extremely accurate Digital Milliwatt Responses specified in the TRANSMISSION PARAMETERS, providing the user a significant margin for error in other board components.

Conversion Laws

The 2913 and 2914 are designed to operate in both μ-law and A-law systems. The user can select either conversion law according to the voltage present on the SIG_X/ASEL pin. In each case the coder and decoder process a companded 8-bit PCM word following CCITT recommendation G.711 for μ-law and A-law conversion. If A-law operation is desired, SIG_X should be tied to V_{BB}. Thus, signaling is not allowed during A-law operation. If μ = 255-law operation is selected, then SIG_X is a TTL level input which modifies the LSB of the PCM output in signaling frames.

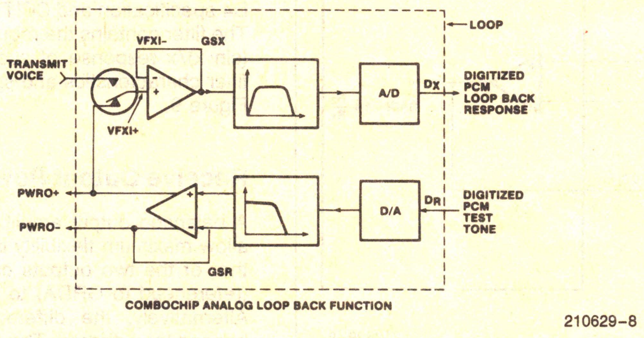


Figure 5. Simplified Block Diagram of 2914 Combochip in the Analog Loopback Configuration

TRANSMIT OPERATION

Transmit Filter

The input section provides gain adjustment in the passband by means of an on-chip operational amplifier. This operational amplifier has a common mode range of ±2.17 volts, a DC offset of 25 mV, an open loop voltage gain of 5000, and a unity gain bandwidth of typically 1 MHz. Gain of up to 20 dB can be set without degrading the performance of the filter. The load impedance to ground (GRDA) at the amplifier output (GS_X) must be greater than 10 KΩ in parallel with less than 50 pF. The input signal on lead VF_XI+ can be either AC or DC coupled. The input op amp can also be used in the inverting mode or differential amplifier mode (see Figure 6).

A low pass anti-aliasing section is included on-chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 channel bank transmission specification and CCITT recommendation G.712. The 2913 and 2914 specifications meet or exceed digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 8.

A high pass section configuration was chosen to reject low frequency noise from 50 Hz and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Even though there is high rejection at these frequencies, the sharpness of the band edge gives low attenuation at 200 Hz. This feature allows the use of low-cost transformer hybrids without external components.

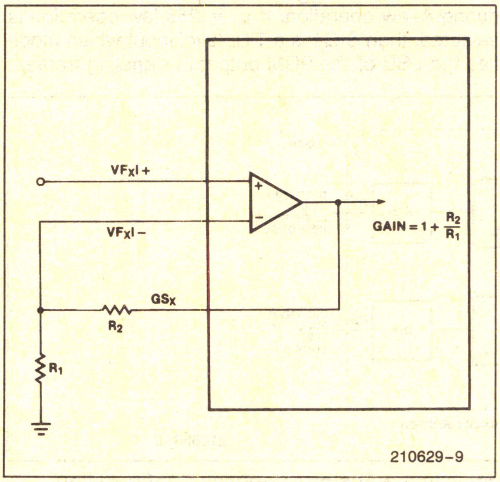


Figure 6. Transmit Filter Gain Adjustment

Encoding

The encoder internally samples the output of the transmit filter and holds each sample on an internal sample and hold capacitor. The encoder then performs an analog to digital conversion on a switched capacitor array. Digital data representing the sample is transmitted on the first eight data clock bits of the next frame.

An on-chip autozero circuit corrects for DC-offset on the input signal to the encoder. This autozero circuit uses the sign bit averaging technique; the sign bit from the encoder output is long term averaged and subtracted from the input to the encoder. In this way, all DC offset is removed from the encoder input waveform.

RECEIVE OPERATION

Decoding

The PCM word at the D_R lead is serially fetched on the first eight data clock bits of the frame. A D/A conversion is performed on the digital word and the corresponding analog sample is held on an internal sample and hold capacitor. This sample is then transferred to the receive filter.

Receive Filter

The receive filter provides passband flatness and stopband rejection which fulfills both the AT&T D3/D4 specification and CCITT recommendation G.712. The filter contains the required compensation for the (sin x)/x response of such decoders. The receive filter characteristics and specifications are shown in Figure 9.

Receive Output Power Amplifiers

A balanced output amplifier is provided in order to allow maximum flexibility in output configuration. Either of the two outputs can be used single ended (referenced to GRDA) to drive single ended loads. Alternatively, the differential output will drive a bridged load directly. The output stage is capable of driving loads as low as 300 ohms single ended or 600 ohms differentially.

Table 4. Zero Transmission Level Points

Symbol	Parameter	Value	Units	Test Conditions
$0TLP1_X$	Zero Transmission Level Point Transmit Channel (0 dBm0) μ-Law	+276 +1.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_X$	Zero Transmission Level Point Transmit Channel (0 dBm0) A-Law	+2.79 +1.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP1_R$	Zero Transmission Level Point Receive Channel (0 dBm0) μ-Law	+5.76 +4.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_R$	Zero Transmission Level Point Receive Channel (0 dBm0) A-Law	+5.79 +4.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω

The receive channel transmission level may be adjusted between specified limits by manipulation of the GS_R input. GS_R is internally connected to an analog gain setting network. When GS_R is strapped to PWRO−, the receive level is maximized; when it is tied to PWRO+, the level is minimized. The output transmission level interpolates between 0 dB and −12 dB as GS_R is interpolated (with a potentiometer) between PWRO+ and PWRO−. The use of the output gain set is illustrated in Figure 7.

Transmission levels are specified relative to the receive channel output under digital milliwatt conditions, that is, when the digital input at D_R is the eight-code sequence specified in CCITT recommendation G.711.

OUTPUT GAIN SET: DESIGN CONSIDERATIONS

(Refer to Figure 7)

PWRO+ and PWRO− are low impedance complementary outputs. The voltages at the nodes are:

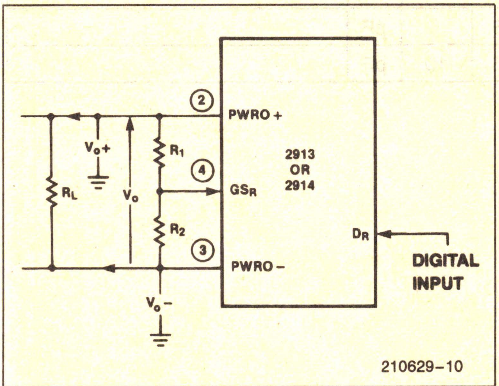


Figure 7. Gain Setting Configuration

V_O+ at PWRO+

V_O- at PWRO−

$V_O = (V_O+) - (V_O-)$(total differential response)

R_1 and R_2 are a gain setting resistor network with the center tap connected to the GS_R input.

A value greater than 10K ohms for $R_1 + R_2$ and less than 100K ohms for R_1 in parallel with R_2 is recommended because:

(a) The parallel combination of $R_1 + R_2$ and R_L sets the total loading.

(b) The total capacitance at the GS_R input and the parallel combination of R_1 and R_2 define a time constant which has to be minimized to avoid inaccuracies.

A is the gain of the power amplifiers, where

$$A = \frac{1 + (R_1/R_2)}{4 + (R_1/R_2)}$$

For design purposes, a useful form is R_1/R_2 as a function of A.

$$R_1/R_2 = \frac{4A - 1}{1 - A}$$

(Allowable values for A are those which make R_1/R_2 positive.)

Examples are:

If A = 1 (maximum output), then

$$R_1/R_2 = \infty \text{ or } V(GS_R) = V_O-;$$

i.e., GS_R is tied to PWRO−

If A = ½, then

$$R_1/R_2 = 2$$

If A = ¼, (minimum output) then

$$R_1/R_2 = 0 \text{ or } V(GS_R) = V_O+;$$

i.e., GS_R is tied to PWRO+.

ABSOLUTE MAXIMUM RATINGS

Temperature under Bias . . . . −10°C to +80°C
Storage Temperature . . . . −65°C to +150°C
V_{CC} and GRDD with Respect to V_{BB} . . . . −0.3V to +15V
All Input and Output Voltages with Respect to V_{BB} . . . . −0.3V to +15V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified

Typical values are for T_A = 25°C and nominal power supply values

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{IL}	Low Level Input Current			10	μA	GRDD ≤ V_{IN} ≤ V_{IL}(1)
I_{IH}	High Level Input Current			10	μA	V_{IH} ≤ V_{IN} ≤ V_{CC}
V_{IL}	Input Low Voltage, except CLKSEL			0.8	V	
V_{IH}	Input High Voltage, except CLKSEL	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 3.2 mA at D_X, $\overline{TS}_X$ and SIG_R
V_{OH}	Output High Voltage	2.4			V	I_{OH} = 9.6 mA at D_X I_{OH} = 1.2 mA at SIG_R
V_{ILO}	Input Low Voltage, CLKSEL(2)	V_{BB}		V_{BB} + 0.5	V	
V_{IIO}	Input Intermediate Voltage, CLKSEL	GRDD − 0.5		0.5	V	
V_{IHO}	Input High Voltage, CLKSEL	V_{CC} − 0.5		V_{CC}	V	
C_{OX}	Digital Output Capacitance(3)		5		pF	
C_{IN}	Digital Input Capacitance		5	10	pF	

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified

Typical values are for T_A = 25°C and nominal power supply values (Continued)

POWER DISSIPATION All measurements made at f_{DCLK} = 2.048 MHz, outputs unloaded

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{CC1}	V_{CC} Operating Current(5)		14	19	mA	
I_{BB1}	V_{BB} Operating Current		−18	−24	mA	
I_{CC0}	V_{CC} Power Down Current		0.5	1.0	mA	$\overline{PDN} \leq V_{IL}$; after 10 μs
I_{BB0}	V_{BB} Power Down Current		−0.5	−1.0	mA	$\overline{PDN} \leq V_{IL}$; after 10 μs
I_{CCS}	V_{CC} Standby Current		1.2	2.4	mA	FS_X, $FS_R \leq V_{IL}$; after 300 ms
I_{BBS}	V_{BB} Standby Current		−1.2	−2.4	mA	FS_X, $FS_R \leq V_{IL}$; after 300 ms
P_{D1}	Operating Power Dissipation(4)		140	200	mW	
P_{D0}	Power Down Dissipation(4)		5	10	mW	$\overline{PDN} \leq V_{IL}$; after 10 μs
P_{ST}	Standby Power Dissipation(4)		12	25	mW	FS_X, $FS_R \leq V_{IL}$

NOTES:
1. V_{IN} is the voltage on any digital pin.
2. SIG_X and $DCLK_R$ are TTL level inputs between GRDD and V_{CC}; they are also pin straps for mode selection when tied to V_{BB}. Under these conditions V_{ILO} is the input low voltage requirement.
3. Timing parameters are guaranteed based on a 100 pF load capacitance. Up to eight digital outputs may be connected to a common PCM highway without buffering, assuming a board capacitance of 60 pF.
4. With nominal power supply values.
5. V_{CC} applied last or simultaneously with V_{BB}.

ANALOG INTERFACE, TRANSMIT CHANNEL INPUT STAGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{BXI}	Input Leakage Current, VF_XI+, VF_XI-			100	nA	$-2.17V \leq V_{IN} \leq 2.17V$
R_{IXI}	Input Resistance, VF_XI+, VF_XI-	10			MΩ	
V_{OSXI}	Input Offset Voltage, VF_XI+, VF_XI-			25	mV	
CMRR	Common Mode Rejection, VF_XI+, VF_XI-	55			dB	$-2.17V \leq V_{IN} \leq 2.17V$
A_{VOL}	DC Open Loop Voltage Gain, GS_X	5000				
f_C	Open Loop Unity Gain Bandwidth, GS_X		1		MHz	
C_{LXI}	Load Capacitance, GS_X			50	pF	
R_{LXI}	Minimum Load Resistance, GS_X	10			KΩ	

ANALOG INTERFACE, RECEIVE CHANNEL DRIVER AMPLIFIER STAGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
R_{ORA}	Output Resistance, PWRO+, PWRO−		1		Ω	
V_{OSRA}	Single-Ended Output DC Offset, PWRO+, PWRO−		75	±150	mV	Relative to GRDA
C_{LRA}	Load Capacitance, PWRO+, PWRO−			100	pF	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

Unless otherwise noted, the analog input is a 0 dBm0, 1020 Hz sine wave.(1) Input amplifier is set for unity gain, noninverting. The digital input is a PCM bit stream generated by passing a 0 dBm0, 1020 Hz sine wave through an ideal encoder. Receive output is measured single ended, maximum gain configuration.(2) All output levels are (sin x)/x corrected. Specifications are for synchronous operation. Typical values are for $T_A = 25°C$ and nominal power supply values. ($T_A = 0°C$ to $+70°C$; $V_{CC} = +5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0V; GRDD = 0V; unless otherwise specified).

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
EmW	Encoder Milliwatt Response (Transmit Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Signal input of 1.064 Vrms μ-law Signal input of 1.068 Vrms A-law $T_A = 25°C$, $V_{BB} = -5V$, $V_{CC} = +5V$
EmW_{TS}	EmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% supplies, 0 to 70°C Relative to nominal conditions
DmW	Digital Milliwatt Response (Receive Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Measure relative to $0TLP_R$. Signal input per CCITT Recommendation G.711. Output signal of 1000 Hz, $R_L = \infty$; $T_A = 25°C$; $V_{BB} = -5V$, $V_{CC} = +5V$.
DmW_{TS}	DmW Variation with Temperature and Supplies	−0.07	+0.02	+0.07	dB	±5% supplies, 0 to 70°C

NOTES:

1. 0 dBm0 is defined as the zero reference point of the channel under test (0TLP). This corresponds to an analog signal input of 1.064 Vrms or an output of 1.503 Vrms for μ-law.
2. Unity gain input amplifier: GS_X is connected to VF_XI-, Signal input VF_XI+; Maximum gain output amplifier; GS_R is connected to PWRO−, output to PWRO+.

GAIN TRACKING Reference Level = −10 dBm0

Symbol	Parameter	2913-1, 2914-1		2913, 2914		Unit	Test Conditions
		Min	Max	Min	Max		
$GT1_X$	Transmit Gain Tracking Error Sinusoidal Input; μ-Law		±0.2 ±0.3 ±0.65		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT2_X$	Transmit Gain Tracking Error Sinusoidal Input; A-Law		±0.2 ±0.3 ±0.65		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT1_R$	Receive Gain Tracking Error Sinusoidal Input; μ-Law		±0.2 ±0.3 ±0.65		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$
$GT2_R$	Receive Gain Tracking Error Sinusoidal Input; A-Law		±0.2 ±0.3 ±0.65		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS (Continued)

NOISE All receive channel measurements are single ended

Symbol	Parameter	2913-1, 2914-1			2913, 2914			Unit	Test Conditions
		Min	Typ	Max	Min	Typ	Max		
N_{XC1}	Transmit Noise, C-Message Weighted			13			15	dBrnc0	VF_XI+ = GRDA, VF_XI- = GS_X
N_{XC2}	Transmit Noise, C-Message Weighted with Eighth Bit Signaling			16			18	dBrnc0	VF_XI+ = GRDA, VF_XI- = GS_X; 6th Frame Signaling
N_{XP}	Transmit Noise, Psophometrically Weighted			−77			−75	dBm0p	VF_XI+ = GRDA, VF_XI- = GS_X
N_{RC1}	Receive Noise, C-Message Weighted: Quiet Code			8			11	dBrnc0	D_R = 11111111
N_{RC2}	Receive Noise, C-Message Weighted: Sign Bit Toggle			9			12	dBrnc0	Input to D_R is zero code with sign bit toggle at 1 KHz rate
N_{RP}	Receive Noise, Psophometrically Weighted			−82			−79	dBm0p	D_R = lowest positive decode level
N_{SF}	Single Frequency Noise End to End Measurement			−50			−50	dBm0	CCITT G.712.4.2, measure at PWRO+
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_2$	V_{BB} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_3$	V_{CC} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle channel; 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz
$PSRR_4$	V_{BB} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle channel; 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz
CT_{TR}	Crosstalk, Transmit to Receive			−80			−71	dB	VF_XI+ = 0 dBm0, 1.02 KHz, D_R = lowest positive decode level, measure at PWRO+
CT_{RT}	Crosstalk, Receive to Transmit			−80			−71	dB	D_R = 0 dBm0, 1.02 KHz VF_XI+ = GRDA, measure at D_X

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS (Continued)

DISTORTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
$SD1_X$	Transmit Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_X$	Transmit Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD1_R$	Receive Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_R$	Receive Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X	Transmit Single Frequency Distortion Products			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
DP_R	Receive Single Frequency Distortion Products			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
IMD_1	Intermodulation Distortion, End to End Measurement			−35	dB	CCITT G.712 (7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−49	dBm0	CCITT G.712 (7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−25	dBm0	CCITT G.712 (6.1)
SIS	Spurious in Band Signals, End to End Measurement			−40	dBm0	CCITT G.712 (9)
D_{AX}	Transmit Absolute Delay		245		μs	Fixed Data Rate. CLK_X = 2.048 MHz 0 dBm0, 1.02 KHz signal at VF_XI+ Measure at D_X.
D_{DX}	Transmit Differential Envelope Delay Relative to D_{AX}		170 95 45 105		μs μs μs μs	f = 500 − 600 Hz f = 600 − 1000 Hz f = 1000 − 2600 Hz f = 2600 − 2800 Hz
D_{AR}	Receive Absolute Delay		190		μs	Fixed Data Rate, CLK_R = 2.048 MHz; Digital input is DMW codes. Measure at PWRO+.
D_{DR}	Receive Differential Envelope Delay Relative to D_{AR}		45 35 85 110		μs μs μs μs	f = 500 − 600 Hz f = 600 − 1000 Hz f = 1000 − 2600 Hz f = 2600 − 2800 Hz

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS (Continued)

TRANSMIT CHANNEL TRANSFER CHARACTERISTICS

Input amplifier is set for unity gain; noninverting; maximum gain output.

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal input at VF_XI+
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−23	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	

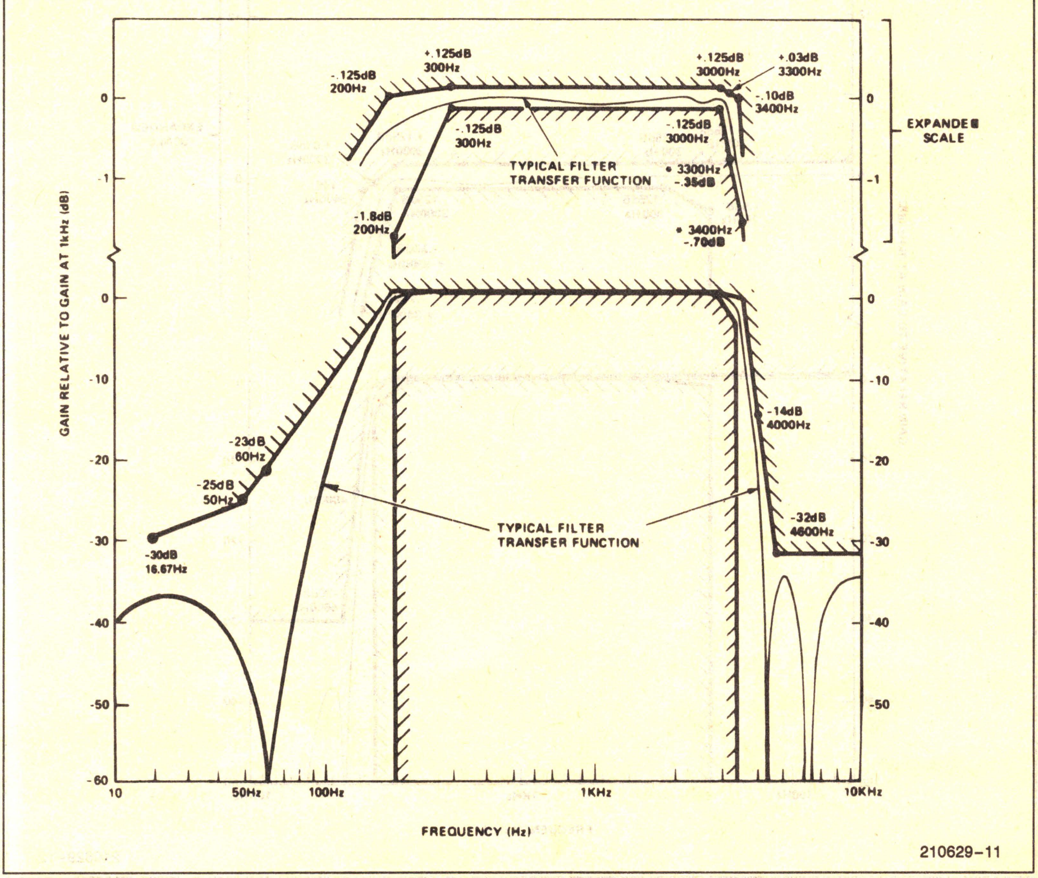


Figure 8. Transmit Channel

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS (Continued)

RECEIVE CHANNEL TRANSFER CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal input at D_R
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−30	dB	

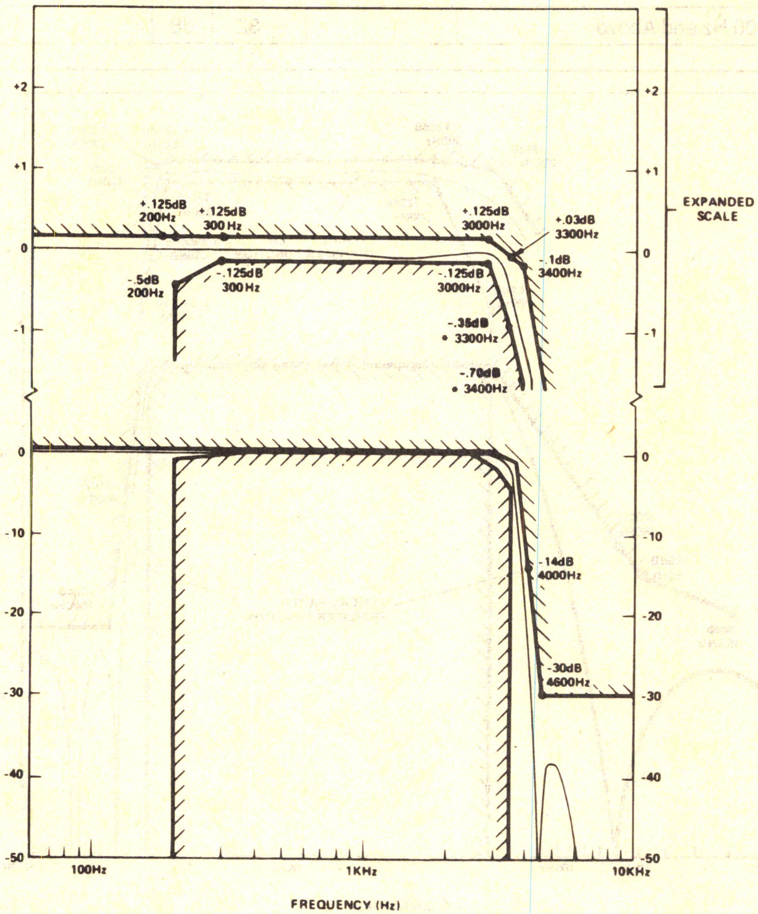


Figure 9. Receive Channel

A.C. CHARACTERISTICS—TIMING PARAMETERS

CLOCK SECTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{CY}	Clock Period, CLK_X, CLK_R	488			ns	$f_{CLKX} = f_{CLKR} = 2.048$ MHz
t_{CLK}	Clock Pulse Width, CLK_X, CLK_R	220			ns	
t_{DCLK}	Data Clock Pulse Width	220			ns	64 KHz $\leq f_{DCLK} \leq$ 2.048 MHz
t_{CDC}	Clock Duty Cycle, CLK_X, CLK_R	45	50	55	%	
t_r, t_f	Clock Rise and Fall Time	5		30	ns	

TRANSMIT SECTION, FIXED DATA RATE MODE(1)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DZX}	Data Enabled on TS Entry	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{DDX}	Data Delay from CLK_X	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	60		215	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{SOFF}	Timeslot X to Disable	60		215	ns	$C_{LOAD} = 0$
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	
t_{SS}	Signal Setup Time	0			ns	
t_{SH}	Signal Hold Time	0			ns	

RECEIVE SECTION, FIXED DATA RATE MODE

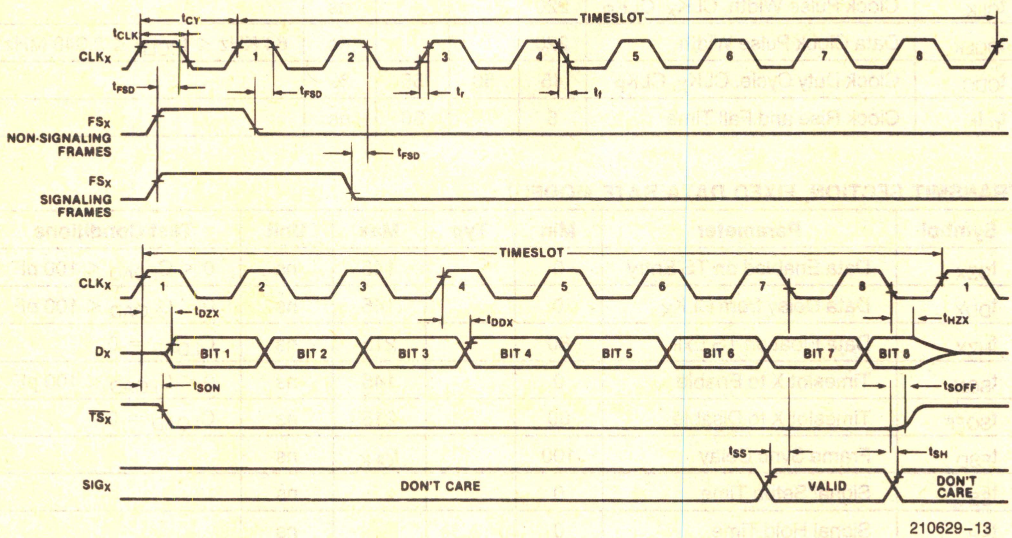
Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DSR}	Receive Data Setup	10			ns	
t_{DHR}	Receive Data Hold	60			ns	
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	
t_{SIGR}	SIG_R Update	0		2	μs	

NOTE:
1. Timing parameters t_{DZX}, t_{HZX}, and t_{SOFF} are referenced to a high impedance state.

WAVEFORMS

Fixed Data Rate Timing

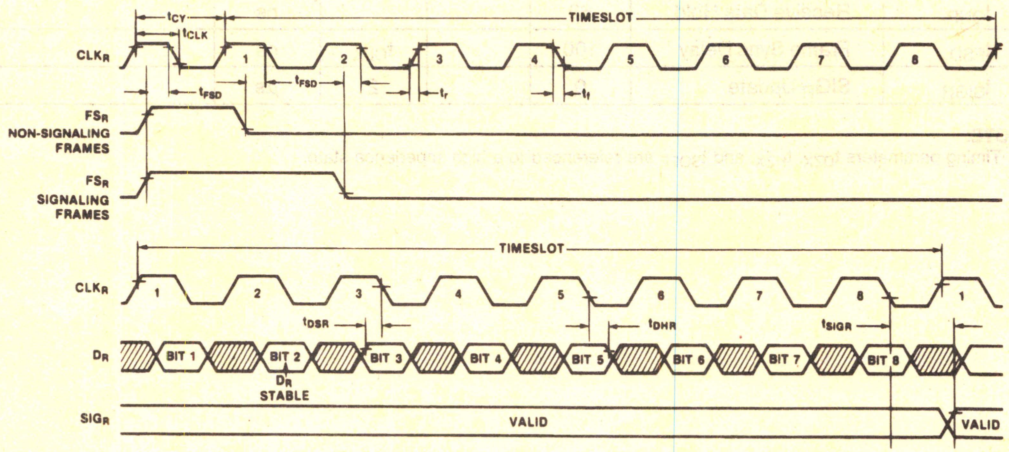
TRANSMIT TIMING



210629–13

NOTE:
All timing parameters referenced to V_{IH} and V_{IL} except t_{DZX}, t_{SOFF} and t_{HZX} which reference a high impedance state.

RECEIVE TIMING



210629–14

NOTE:
All timing parameters referenced to V_{IH} and V_{IL}.

WAVEFORMS (Continued)

TRANSMIT SECTION, VARIABLE DATA RATE MODE(1)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDX}	Timeslot Delay from $DCLK_X$(2)	140		$t_{DX} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DDX}	Data Delay from $DCLK_X$	0		100	ns	$0 < C_{LOAD} < 100$ pF
t_{DON}	Timeslot to D_X Active	0		50	ns	$0 < C_{LOAD} < 100$ pF
t_{DOFF}	Timeslot to D_X Inactive	0		80	ns	$0 < C_{LOAD} < 100$ pF
t_{DX}	Data Clock Period	488		15620	ns	
t_{DFSX}	Data Delay from FS_X	0		140	ns	

RECEIVE SECTION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDR}	Timeslot Delay from $DCLK_R$(3)	140		$t_{DR} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DSR}	Data Setup Time	10			ns	
t_{DHR}	Data Hold Time	60			ns	
t_{DR}	Data Clock Period	488		15620	ns	
t_{SER}	Timeslot End Receive Time	60			ns	

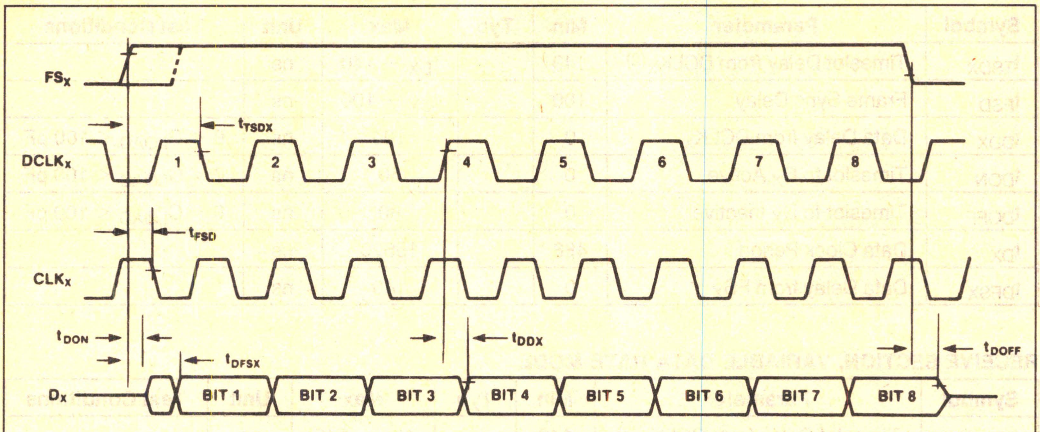
64 KB OPERATION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{FSLX}	Transmit Frame Sync Minimum Downtime	488			ns	FS_X is TTL high for remainder of frame
t_{FSLR}	Receive Frame Sync Minimum Downtime	1952			ns	FS_R is TTL high for remainder of frame
t_{DCLK}	Data Clock Pulse Width			10	μs	

NOTES:
1. Timing parameters for t_{DON} and t_{DOFF} are referenced to a high impedance state.
2. t_{FSLX} minimum requirements override t_{TSDX} maximum spec for 64 KHz operation.
3. t_{FSLR} minimum requirements override t_{TSDR} maximum spec for 64 KHz operation.

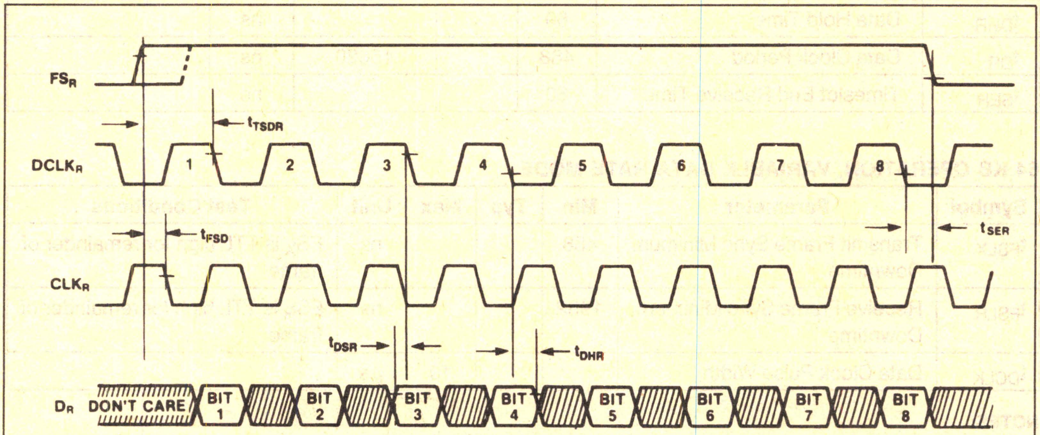
VARIABLE DATA RATE TIMING

TRANSMIT TIMING



210629-15

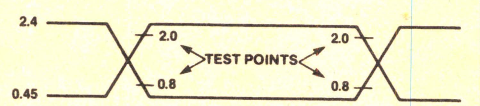
RECEIVE TIMING



210629-16

NOTE:
All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{DOFF} which reference a high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM



210629-17

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

29C13 AND 29C14 CHMOS COMBINED SINGLE-CHIP PCM CODEC AND FILTER

- 29C14 Asynchronous Clocks, 8th Bit Signaling, Loop Back Test Capability
- 29C13 Synchronous Clocks Only, 300 Mil Package
- Low-Power Pin Compatible Version of Intel's 2913 and 2914
- 28-Pin Plastic Leaded Chip Carrier (PLCC) for Higher Integration
- AT&T D3/D4 and CCITT Compatible
- 3 Low-Power Modes
  - —5 mW Typical Power Down
  - —8 mW Typical Standby
  - —70 mW Typical Operating
- Direct Interface with Transformer or Electronic Hybrids
- TTL and CMOS Compatible

Intel's 29C13 and 29C14 are CHMOS versions of Intel's HMOS 2913 and 2914 family members. CHMOS is a technology built on HMOS-II, thus realizing the high performance and density obtained in that process while achieving the low power consumption typical of CMOS circuits.

The 29C13 and 29C14 retain all the features of the 2913 and 2914: push/pull power amplifiers, μ/A law pin select, on-chip auto zero, sample and hold and precision voltage references, power up clear and tri-state on clock interrupt, two timing modes and two power down modes.

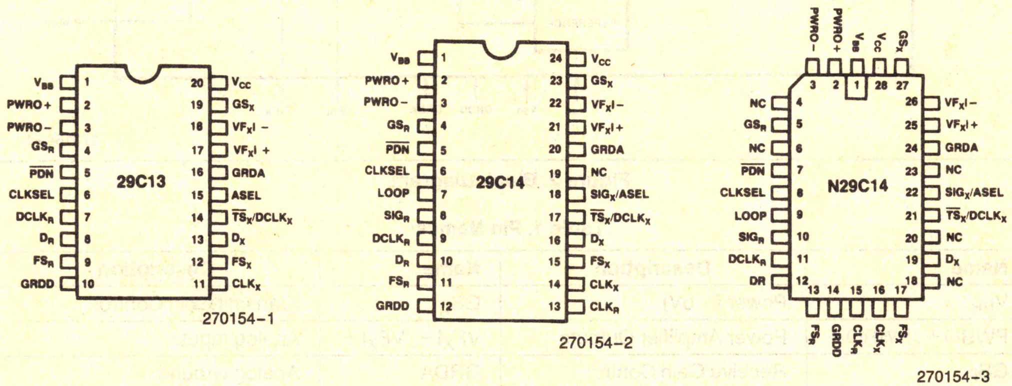


Figure 1. Pin Configurations

November 1986
Order Number: 270154-001

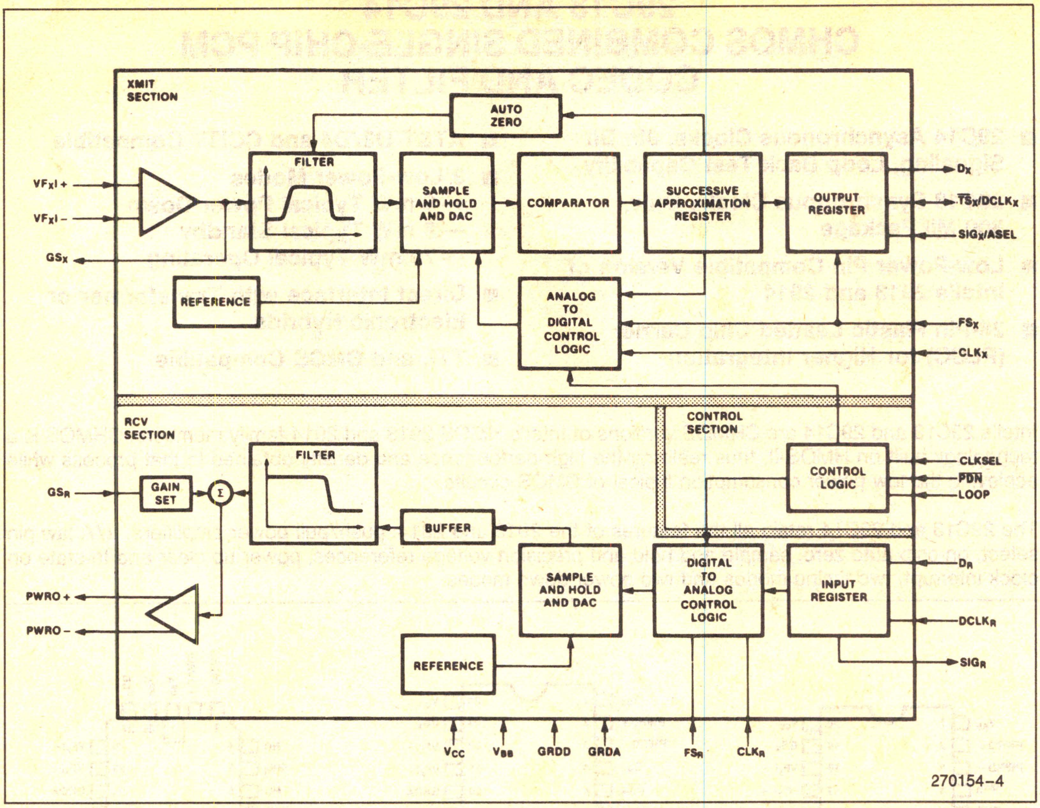


Figure 2. Block Diagram

Table 1. Pin Names

Name	Description	Name	Description
V_{BB}	Power (−5V)	GS_X	Transmit Gain Control
PWRO+, PWRO−	Power Amplifier Outputs	VF_XI-, VF_XI+	Analog Inputs
GS_R	Receive Gain Control	GRDA	Analog Ground
$\overline{PDN}$	Power Down Select	NC	No Connect
CLKSEL	Master Clock Frequency Select	SIG_X	Transmit Signaling Input
LOOP	Analog Loop Back	ASEL	μ- or A-law Select
SIG_R	Receive Signaling Bit Output	$\overline{TS}_X$	Timeslot Strobe/Buffer Enable
$DCLK_R$	Receive Variable Data Clock	$DCLK_X$	Transmit Variable Data Clock
D_R	Receive PCM Input	D_X	Transmit PCM Output
FS_R	Receive Frame Synchronization Clock	FS_X	Transmit Frame Synchronization Clock
GRDD	Digital Ground	CLK_X	Transmit Master Clock
V_{CC}	Power (+5V)	CLK_R	Receive Master Clock (29C14 only, internally connected to CLK_X on 29C13)

Table 2. Pin Description

Symbol	Function
V_{BB}	Most negative supply; input voltage is −5 volts ±5%.
PWRO+	Non-inverting output of power amplifier. Can drive transformer hybrids or high impedance loads directly in either a differential or single ended configuration.
PWRO−	Inverting output of power amplifier. Functionally identical and complementary to PWRO+.
GS_R	Input to the gain setting network on the output power amplifier. Transmission level can be adjusted over a 12 dB range depending on the voltage at GS_R.
$\overline{PDN}$	Power down select. When $\overline{PDN}$ is TTL high, the device is active. When low, the device is powered down.
CLKSEL	Input which must be pinstrapped to reflect the master clock frequency at CLK_X, CLK_R. CLKSEL = V_{BB} .... 2.048 MHz CLKSEL = GRDD .... 1.544 MHz CLKSEL = V_{CC} .... 1.536 MHz
LOOP	Analog loopback. When this pin is TTL high, the analog output (PWRO+) is internally connected to the analog input (VF_XI+), GS_R is internally connected to PWRO−, and VF_XI- is internally connected to GS_X. A 0 dBm0 digital signal input at D_R is returned as a +3 dBm0 digital signal output at D_X.
SIG_R	Signaling bit output, receive channel. In fixed data rate mode, SIG_R outputs the logical state of the eighth bit of the PCM word in the most recent signaling frame.
$DCLK_R$	Selects the fixed or variable data rate mode. When $DCLK_R$ is connected to V_{BB}, the fixed data rate mode is selected. In this mode, the device is fully compatible with Intel 2910A and 2911A direct mode timing. When $DCLK_R$ is not connected to V_{BB}, the device operates in the variable data rate mode. In this mode $DCLK_R$ becomes the receive data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
D_R	Receive PCM input. PCM data is clocked in on this lead on eight consecutive negative transitions of the receive data clock; CLK_R in the fixed data rate mode and $DCLK_R$ in variable data rate mode.
FS_R	8 KHz frame synchronization clock input/timeslot enable, receive channel. A multi-function input which in fixed data rate mode distinguishes between signaling and non-signaling frames by means of a double or single wide pulse respectively. In variable data rate mode this signal must remain high for the entire length of the timeslot. The receive channel enters the standby state whenever FS_R is TTL low for 300 milliseconds.
GRDD	Digital ground for all internal logic circuits. Not internally tied to GRDA.
CLK_R	Receive master and data clock for the fixed data rate mode; receive master clock only in variable data rate mode.
CLK_X	Transmit master and data clock for the fixed data rate mode; transmit master clock only in variable data rate mode.
FS_X	8 KHz frame synchronization clock input/timeslot enable, transmit channel. Operates independently but in an analogous manner to FS_R. The transmit channel enters the standby state whenever FS_X is TTL low for 300 milliseconds.
D_X	Transmit PCM output. PCM data is clocked out on this lead on eight consecutive positive transitions of the transmit data clock: CLK_X in fixed data rate mode and $DCLK_X$ in variable data rate mode.
$\overline{TS}_X/DCLK_X$	Transmit channel timeslot strobe (output) or data clock (input) for the transmit channel. In fixed data rate mode, this pin is an open drain output designed to be used as an enable signal for a three-state buffer as in 2910A and 2911A direct mode timing. In variable data rate mode, this pin becomes the transmit data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
SIG_X/ASEL	A dual purpose pin. When connected to V_{BB}, A-law operation is selected. When it is not connected to V_{BB} this pin is a TTL level input for signaling operation. This input is transmitted as the eighth bit of the PCM word during signaling frames on the D_X lead. If not used as an input pin, ASEL should be strapped to either V_{CC} or GRDD.
NC	No Connect
GRDA	Analog ground return for all internal voice circuits. Not internally connected to GRDD.
VF_XI+	Non-inverting analog input to uncommitted transmit operational amplifier.
VF_XI-	Inverting analog input to uncommitted transmit operational amplifier.
GS_X	Output terminal of transmit input channel op amp. Internally, this is the voice signal input to the transmit filter.
V_{CC}	Most positive supply; input voltage is +5 volts ±5%.

FUNCTIONAL DESCRIPTION

The 2913 and 2914 provide the analog-to-digital and the digital-to-analog conversions and the transmit and receive filtering necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. They are intended to be used at the analog termination of a PCM line or trunk.

The following major functions are provided:

- Bandpass filtering of the analog signals prior to encoding and after decoding
- Encoding and decoding of voice and call progress information
- Encoding and decoding of the signaling and supervision information

SWITCHING

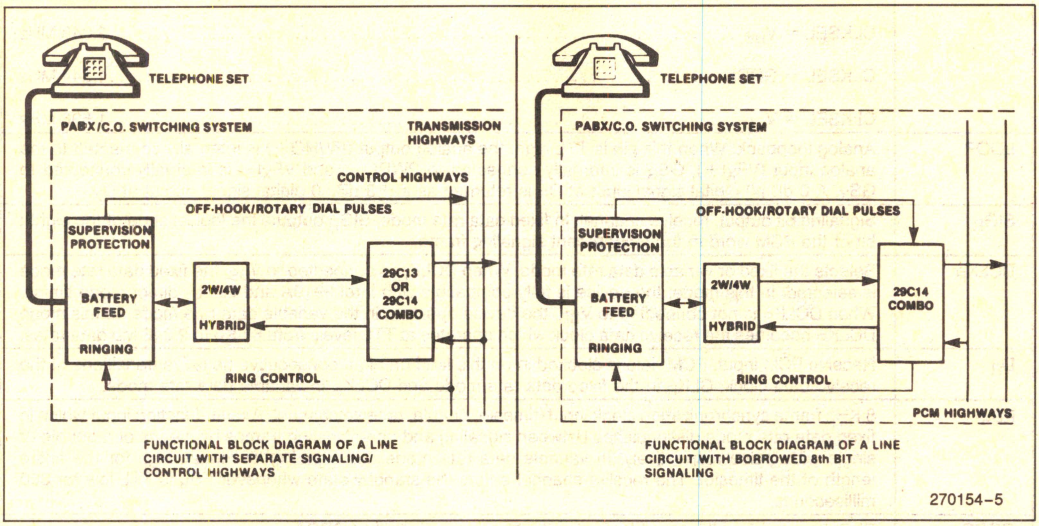


Figure 3a. Typical Line Terminations

CHANNEL BANKS

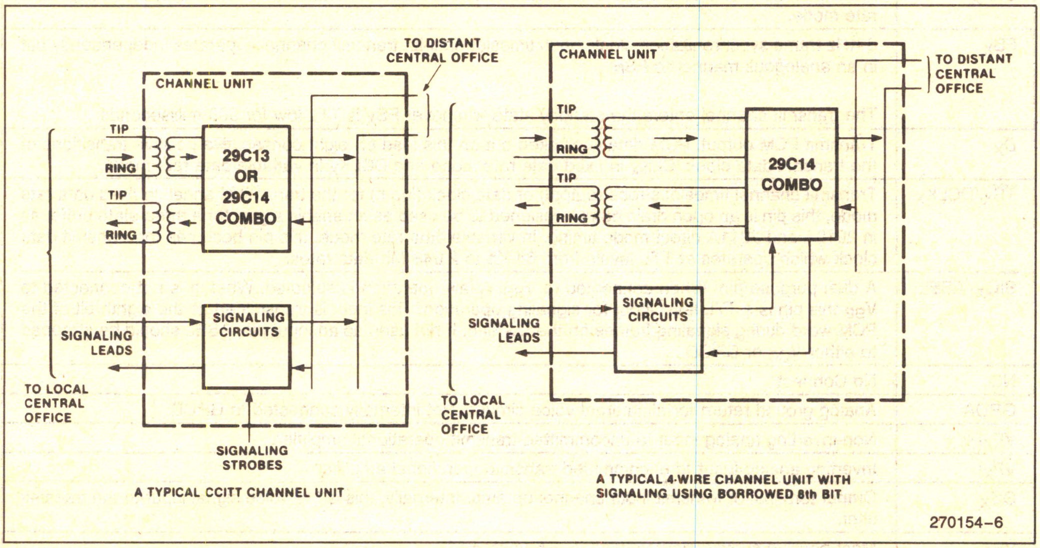


Figure 3b. Typical Line Terminations

GENERAL OPERATION

System Reliability Features

The combochip can be powered up by pulsing FS_X and/or FS_R while a TTL high voltage is applied to $\overline{PDN}$, provided that all clocks and supplies are connected. The 29C13 and 29C14 have internal resets on power up (or when V_{BB} or V_{CC} are re-applied) in order to ensure validity of the digital outputs and thereby maintain integrity of the PCM highway.

On the transmit channel, digital outputs D_X and $\overline{TS_X}$ are held in a high impedance state for approximately four frames (500 μs) after power up or application of V_{BB} or V_{CC}. After this delay, D_X, $\overline{TS_X}$, and signaling will be functional and will occur in the proper time-slot. The analog circuits on the transmit side require approximately 60 milliseconds to reach their equilibrium value due to the autozero circuit settling time. Thus, valid digital information, such as for on/off hook detection, is available almost immediately, while analog information is available after some delay.

On the receive channel, the digital output SIG_R is also held low for a maximum of four frames after power up or application of V_{BB} or V_{CC}. SIG_R will remain low thereafter until it is updated by a signaling frame.

To further enhance system reliability, $\overline{TS_X}$ and D_X will be placed in a high impedance state approximately 30 μs after an interruption of CLK_X. Similarly, SIG_R will be held low approximately 30 μs after an interruption of CLK_R. These interruptions could possibly occur with some kind of fault condition.

Power Down and Standby Modes

To minimize power consumption, two power down modes are provided in which most 29C13/C14 functions are disabled. Only the power down, clock, and frame sync buffers, which are required to power up the device, are enabled in these modes. As shown in Table 3, the digital outputs on the appropriate channels are placed in a high impedance state until the device returns to the active mode.

The Power Down mode utilizes an external control signal to the $\overline{PDN}$ pin. In this mode, power consumption is reduced to the value shown in Table 3. The device is active when the signal is high and inactive when it is low. In the absence of any signal, the $\overline{PDN}$ pin floats to TTL high allowing the device to remain active continuously.

The Standby mode leaves the user an option of powering either channel down separately or powering the entire device down by selectively removing FS_X and/or FS_R. With both channels in the standby state, power consumption is reduced to the value shown in Table 3. If transmit only operation is desired, FS_X should be applied to the device while FS_R is held low. Similarly, if receive only operation is desired, FS_R should be applied while FS_X is held low.

Fixed Data Rate Mode

Fixed data rate timing, which is 2910A and 2911A compatible, is selected by connecting $DCLK_R$ to V_{BB}. It employs master clocks CLK_X and CLK_R, frame synchronization clocks FS_X and FS_R, and output $\overline{TS_X}$.

Table 3. Power-Down Methods

Device Status	Power-Down Method	Typical Power Consumption	Digital Output Status
Power Down Mode	$\overline{PDN}$ = TTL low	5 mW	$\overline{TS_X}$ and D_X are placed in a high impedance state and SIG_R is placed in a TTL low state within 10 μs.
Standby Mode	FS_X and FS_R are TTL low	8 mW	$\overline{TS_X}$ and D_X are placed in a high impedance state and SIG_R is placed in a TTL low state 300 milliseconds after FS_X and FS_R are removed.
Only transmit is on Standby	FS_X is TTL low	50 mW	$\overline{TS_X}$ and D_X are placed in a high impedance state within 300 milliseconds.
Only receive is on Standby	FS_R is TTL low	50 mW	SIG_R is placed in a TTL low state within 300 milliseconds.

CLK_X and CLK_R serve both as master clocks to operate the codec and filter sections and bit clocks to clock the data in and out from the PCM highway. FS_X and FS_R are 8 KHz inputs which set the sampling frequency and distinguish between signaling and non-signaling frames by their pulse width. A frame synchronization pulse which is one master clock wide designates a non-signaling frame, while a double wide sync pulse enables the signaling function. $\overline{TS}_X$ is a timeslot strobe/buffer enable output which gates the PCM word onto the PCM highway when an external buffer is used to drive the line.

Data is transmitted on the highway at D_X on the first eight positive transitions of CLK_X following the rising edge of FS_X. Similarly, on the receive side, data is received on the first eight falling edges of CLK_R. The frequency of CLK_X and CLK_R is selected by the CLKSEL pin to be either 1.536, 1.544, or 2.048 MHz. No other frequency of operation is allowed in the fixed data rate mode.

Variable Data Rate Mode

Variable data rate timing is selected by connecting $DCLK_R$ to the bit clock for the receive PCM highway rather than to V_{BB}. It employs master clocks CLK_X and CLK_R, bit clocks $DCLK_R$ and $DCLK_X$, and frame synchronization clocks FS_R and FS_X.

Variable data rate timing allows for a flexible data frequency. It provides the ability to vary the frequency of the bit clocks, which can be asynchronous in the case of the 29C14, synchronous in the case of the 29C13, from 64 KHz to 2.048 MHz. Master clocks inputs are still restricted to 1.536, 1.544, or 2.048 MHz.

In this mode, $DCLK_R$ and $DCLK_X$ become the data clocks for the receive and transmit PCM highways. While FS_X is high, PCM data from D_X is transmitted onto the highway on the next eight consecutive positive transitions of $DCLK_X$. Similarly, while FS_R is high, each PCM bit from the highway is received by D_R on the next eight consecutive negative transitions of $DCLK_R$.

On the transmit side, the PCM word will be repeated in all remaining timeslots in the 125 μs frame as long as $DCLK_X$ is pulsed and FS_X is held high. This feature allows the PCM word to be transmitted to the PCM highway more than once per frame, if desired, and is only available in the variable data rate mode. Conversely, signaling is only allowed in the fixed data rate mode since the variable mode provides no means with which to specify a signaling frame.

Signaling

Signaling can only be performed with the 24-pin device in the fixed data rate timing mode ($DCLK_R$ = V_{BB}). Signaling frames on the transmit and receive sides are independent of one another and are selected by a double-width frame sync pulse on the appropriate channel. During a transmit signaling frame, the codec will encode the incoming analog signal and substitute the signal present on SIG_X for the least significant bit of the encoded PCM word. Similarly, in a receive signaling frame, the codec will decode the seven most significant bits according to CCITT recommendation G.733 and output the logical state of the LSB on the SIG_R lead until it is updated in the next signaling frame. Timing relationships for signaling operation are shown in Figure 4.

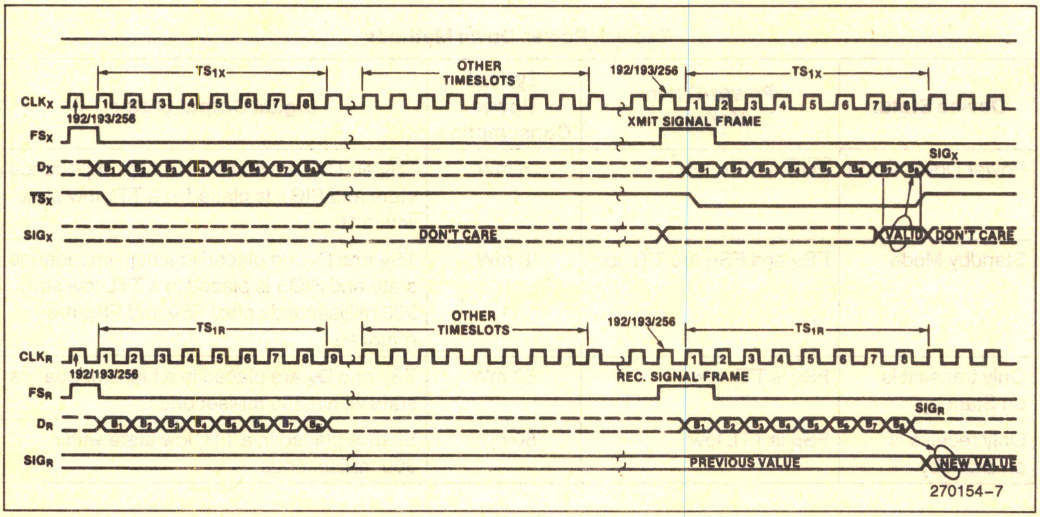


Figure 4. Signaling Timing (Used Only with Fixed Data Rate Mode)

Asynchronous Operation

The 29C14 can be operated with asynchronous clocks in either the fixed or variable data rate modes. In order to avoid crosstalk problems associated with special interrupt circuitry, the design of the Intel 29C13/C14 combochip includes separate digital-to-analog converters and voltage references on the transmit and receive sides to allow independent operation of the two channels.

In either timing mode, the master clock, data clock, and timeslot strobe must be synchronized at the beginning of each frame. CLK_X and $DCLK_X$ are synchronized once per frame but may be of different frequencies. The receive channel operates in a similar manner and is completely independent of the transmit timing (refer to Variable Data Rate Timing Diagrams). This approach requires the provision of two separate master clocks, even in variable data rate mode, but avoids the use of a synchronizer which can cause intermittent data conversion errors.

Analog Loopback

A distinctive feature of the 29C14 is its analog loopback capability. This feature allows the user to send a control signal which internally connects the analog input and output ports. As shown in Figure 5, when LOOP is TTL high the analog output (PWRO+) is internally connected to the analog input (VF_XI+), GS_R is internally connected to PWRO−, and VF_XI- is internally connected to GS_X.

With this feature, the user can test the line circuit remotely by comparing the digital codes sent into the receive channel (D_R) with those generated on the transmit channel (D_X). Due to the difference in transmission levels between the transmit and receive sides, a 0 dBm0 code sent into D_R will emerge from D_X as a +3 dBm0 code, an implicit gain of 3 dB. Thus, the maximum signal input level which can be tested using analog loopback is 0 dBm0.

Precision Voltage References

No external components are required with the combochip to provide the voltage reference function. Voltage references are generated on-chip and are calibrated during the manufacturing process. These references determine the gain and dynamic range characteristics of the device.

Separate references are supplied to the transmit and receive sections and each is trimmed independently during the manufacturing process. The reference value is then further trimmed in the gain setting op-amps to a final precision value. With this method the combochip can achieve the extremely accurate Digital Milliwatt Responses specified in the TRANSMISSION PARAMETERS, providing the user a significant margin for error in other board components.

Conversion Laws

The 29C13 and 29C14 are designed to operate in both μ-law and A-law systems. The user can select either conversion law according to the voltage present on the SIG_X/ASEL pin. In each case the coder and decoder process a companded 8-bit PCM word following CCITT recommendation G.711 for μ-law and A-law conversion. If A-law operation is desired, SIG_X should be tied to V_{BB}. Thus, signaling is not allowed during A-law operation. If μ = 255-law operation is selected, then SIG_X is a TTL level input which modifies the LSB of the PCM output in signaling frames.

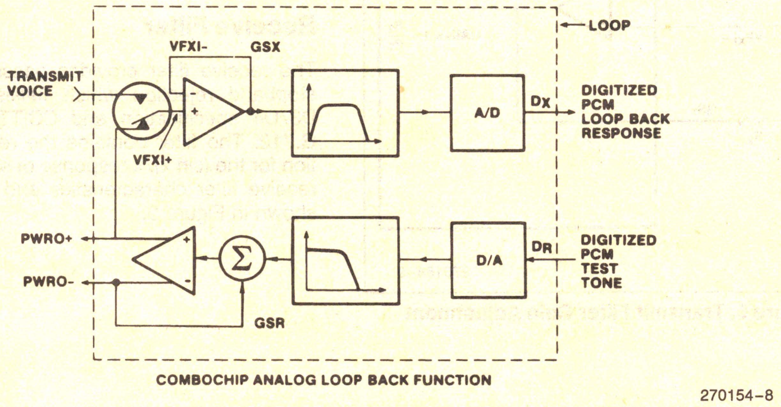


Figure 5. Simplified Block Diagram of 29C14 Combochip in the Analog Loopback Configuration

TRANSMIT OPERATION

Transmit Filter

The input section provides gain adjustment in the passband by means of an on-chip uncommitted operational amplifier. This operational amplifier has a common mode range of ±2.17 volts, a DC offset of 25 mV, and a typical voltage gain of 20,000. Gain of up to 20 dB can be set without degrading the performance of the filter. The load impedance to ground (GRDA) at the amplifier output (GS_X) must be greater than 10 kilohms in parallel with less than 50 pF. A DC path must be provided at VF_XI+. The input op amp can also be used in the inverting mode or differential amplifier mode (see Figure 6).

A low pass anti-aliasing section is included on-chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 channel bank transmission specification and CCITT recommendation G.712. The 29C13 and 29C14 specifications meet or exceed digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 8.

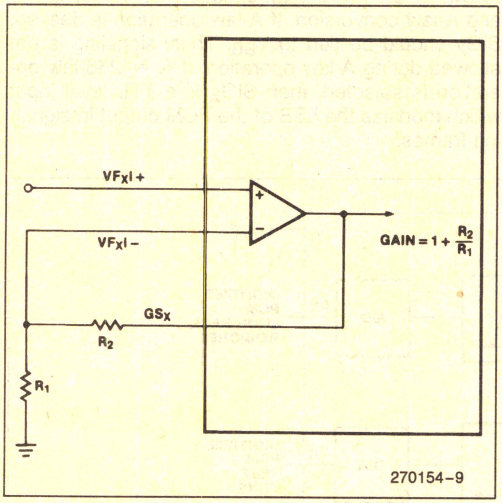


Figure 6. Transmit Filter Gain Adjustment

A high pass section configuration was chosen to reject low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Even though there is high rejection at these frequencies, the sharpness of the band edge gives low attenuation at 200 Hz. This feature allows the use of low-cost transformer hybrids without external components.

Encoding

The encoder internally samples the output of the transmit filter and holds each sample on an internal sample and hold capacitor. The encoder then performs an analog to digital conversion on a switched capacitor array. Digital data representing the sample is transmitted on the first eight data clock bits of the next frame.

An on-chip autozero circuit corrects for DC-offset on the input signal to the encoder. This autozero circuit uses the sign bit averaging technique; the sign bit from the encoder output is long term averaged and subtracted from the input to the encoder. In this way, all DC offset is removed from the encoder input waveform.

RECEIVE OPERATION

Decoding

The PCM word at the D_R lead is serially fetched on the first eight data clock bits of the frame. A D/A conversion is performed on the digital word and the corresponding analog sample is held on an internal sample and hold capacitor. This sample is then transferred to the receive filter.

Receive Filter

The receive filter provides passband flatness and stopband rejection which fulfills both the AT&T D3/D4 specification and CCITT recommendation G.712. The filter contains the required compensation for the (sin x)/x response of such decoders. The receive filter characteristics and specifications are shown in Figure 9.

Table 4. Zero Transmission Level Points

Symbol	Parameter	Value	Units	Test Conditions
$0TLP1_X$	Zero Transmission Level Point Transmit Channel (0 dBm0) μ-law	+2.76 +1.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_X$	Zero Transmission Level Point Transmit Channel (0 dBm0) A-law	+2.79 +1.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP1_R$	Zero Transmission Level Point Receive Channel (0 dBm0) μ-law	+5.76 +4.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_R$	Zero Transmission Level Point Receive Channel (0 dBm0) A-law	+5.79 +4.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω

Receive Output Power Amplifiers

A balanced output amplifier is provided in order to allow maximum flexibility in output configuration. Either of the two outputs can be used single ended (referenced to GRDA) to drive single ended loads. Alternatively, the differential output will drive a bridged load directly. The output stage is capable of driving loads as low as 300 ohms single ended or 600 ohms differentially.

The receive channel transmission level may be adjusted between specified limits by manipulation of the GS_R input. GS_R is internally connected to an analog gain setting network. When GS_R is strapped to PWRO−, the receive level is unattenuated; when it is tied to PWRO+, the level is attenuated by 12 dB. The output transmission level interpolates between 0 and −12 dB as GS_R is interpolated (with a potentiometer) between PWRO+ and PWRO−. The use of the output gain set is illustrated in Figure 7.

Transmission levels are specified relative to the receive channel output under digital milliwatt conditions, that is, when the digital input at D_R is the eight-code sequence specified in CCITT recommendation G.711.

OUTPUT GAIN SET: DESIGN CONSIDERATIONS

(Refer to Figure 7.)

PWRO+ and PWRO− are low impedance complementary outputs. The voltages at the nodes are:

V_O+ at PWRO+
V_O- at PWRO−

$V_O = (V_O+) - (V_O-)$ (total differential response)

R_1 and R_2 are a gain setting resistor network with the center tap connected to the GS_R input.

A value greater than 10K ohms for $R_1 + R_2$ and less than 100K ohms for R_1 in parallel with R_2 is recommended because:

(a) The parallel combination of $R_1 + R_2$ and R_L sets the total loading.

(b) The total capacitance at the GS_R input and the parallel combination of R_1 and R_2 define a time constant which has to be minimized to avoid inaccuracies.

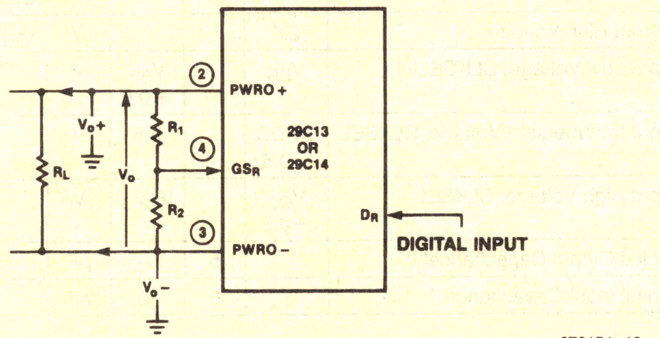


Figure 7. Gain Setting Configuration

A is the gain of the power amplifiers,

$$\text{where } A = \frac{1 + (R_1/R_2)}{4 + (R_1/R_2)}$$

For design purposes, a useful form is R_1/R_2 as a function of A.

$$R_1/R_2 = \frac{4A - 1}{1 - A}$$

(Allowable values for A are those which make R_1/R_2 positive.)

Examples are:

If A = 1 (maximum output), then

$R_1/R_2 = \infty$ or $V(GS_R) = V_O-$; i.e., GS_R is tied to PWRO−

If A = ½, then

$R_1/R_2 = 2$

If A = ¼, (minimum output) then

$R_1/R_2 = 0$ or $V(GS_R) = V_O+$; i.e., GS_R is tied to PWRO+

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
V_{CC} and GRDD with Respect to V_{BB} .. −0.3V to 15V
All Input and Output Voltages with Respect to V_{BB} .... −0.3V to 15V
All Input and Output Voltages with Respect to V_{CC} .... −15V to +0.3V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

(T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified.) Typical values are for T_A = 25°C and nominal power supply values.

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{IL}	Low Level Input Current			10	μA	GRDD ≤ V_{IN} ≤ V_{IL} (Note 1)
I_{IH}	High Level Input Current			10	μA	V_{IH} ≤ V_{IN} ≤ V_{CC}
V_{IL}	Input Low Voltage, except CLKSEL			0.8	V	
V_{IH}	Input High Voltage, except CLKSEL	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 3.2 mA at D_X, $\overline{TS}_X$ and SIG_R
V_{OH}	Output High Voltage	2.4			V	I_{OH} = 80 μA at D_X, SIG_R
V_{ILO}	Input Low Voltage, CLKSEL(2)	V_{BB}		V_{BB} +0.5	V	
V_{IIO}	Input Intermediate Voltage, CLKSEL	GRDD −0.5		0.5	V	
V_{IHO}	Input High Voltage, CLKSEL	V_{CC} −0.5		V_CV_C	V	
C_{OX}	Digital Output Capacitance(3)		5		pF	
C_{IN}	Digital Input Capacitance		5	10	pF	

POWER DISSIPATION

All measurements made at f_{DCLK} = 2.048 MHz, outputs unloaded.

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{CCI}	V_{CC} Operating Current		5.6		mA	
I_{BBI}	V_{BB} Operating Current		−5.6		mA	
I_{CCO}	V_{CC} Power Down Current		0.5		mA	$\overline{PDN} \leq V_{IL}$; after 10 μs
I_{BBO}	V_{BB} Power Down Current		−0.5		mA	$\overline{PDN} \leq V_{IL}$; after 10 μs
I_XC_{CS}	V_{CC} Standby Current		0.8		mA	FS_X, $FS_R \leq V_{IL}$; after 300 ms
I_{BBS}	V_{BB} Standby Current		−0.8		mA	FS_X, $FS_R \leq V_{IL}$; after 300 ms
P_{DI}	Operating Power Dissipation(4)		70		mW	
P_{DO}	Power Down Dissipation(4)		5		mW	$\overline{PDN} \leq V_{IL}$; after 10 μs
P_{ST}	Standby Power Dissipation(4)		8		mW	FS_X, $FS_R \leq V_{IL}$

NOTES:
1. V_{IN} is the voltage on any digital pin.
2. SIG_X and $DCLK_R$ are TTL level inputs between GRDD and V_{CC}; they are also pinstraps for mode selection when tied to V_{BB}. Under these conditions V_{ILO} is the input low voltage requirement.
3. Timing parameters are guaranteed based on a 100 pF load capacitance. Up to eight digital outputs may be connected to a common PCM highway without buffering, assuming a board capacitance of 60 pF.
4. With nominal power supply values.

ANALOG INTERFACE, TRANSMIT CHANNEL INPUT STAGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{BX1}	Input Leakage Current, VF_XI+, VF_XI-			100	nA	$-2.17V \leq V_{IN} \leq 2.17V$
R_{IXI}	Input Resistance, VF_XI+, VF_XI-	10			MΩ	
V_{OSXI}	Input Offset Voltage, VF_XI+, VF_XI-			25	mV	
CMRR	Common Mode Rejection, VF_XI+, VF_XI-	55			dB	$-2.17 \leq V_{IN} \leq 2.17V$
A_{VOL}	DC Open Loop Voltage Gain, GS_X	5000				
f_C	Open Loop Unity Gain Bandwidth, GS_X		1		MHz	
C_{LXI}	Load Capacitance, GS_X			50	pF	
R_{LXI}	Minimum Load Resistance, GS_X	10			KΩ	

ANALOG INTERFACE, RECEIVE CHANNEL DRIVER AMPLIFIER STAGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
R_{ORA}	Output Resistance, PWRO+, PWRO−		1		Ω	
V_{OSRA}	Single-Ended Output DC Offset, PWRO+ PWRO−		75	±150	mV	Relative to GRDA
C_{LRA}	Load Capacitance, PWRO+, PWRO−			100	pF	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

Unless otherwise noted, the analog input is a 0 dBm0, 1020 Hz sine wave.(1) Input amplifier is set for unity gain, noninverting. The digital input is a PCM bit stream generated by passing a 0 dBm0, 1020 Hz sine wave through an ideal encoder. Receive output is measured single ended, maximum gain configuration.(2) All output levels are (sin x)/x corrected.

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response (Transmit Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Signal input of 1.064 Vrms μ-law Signal input of 1.068 Vrms A-law $T_A = 25°C$, $V_{BB} = -5V$, $V_{CC} = +5V$
EmW_{TS}	EmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% supplies, 0 to 70°C Relative to nominal conditions
DmW	Digital Milliwatt Response (Receive Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Measure relative to $0TLP_R$. Signal input per CCITT Recommendation G.711. Output signal of 1000 Hz. $T_A = 25°C$; $V_{BB} = -5V$, $V_{CC} = +5V$. $R_L = \infty$
DmW_{TS}	DmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% supplies, 0 to 70°C

NOTES:
1. 0 dBm0 is defined as the zero reference point of the channel under test (0TLP). This corresponds to an analog signal input of 1.064 volts rms or an output of 1.503 volts rms for μlaw.
2. Unity gain input amplifier: GS_X is connected to VF_XI-, Signal input VF_XI+; Maximum gain output amplifier; GS_R is connected to PWRO−, output to PWRO+.

GAIN TRACKING

Reference Level = −10 dBm0

Symbol	Parameter	Min	Max	Unit	Test Conditions
$GT1_X$	Transmit Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT2_X$	Transmit Gain Tracking Error Sinusoidal Input; A-law		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT1_R$	Receive Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$
$GT2_R$	Receive Gain Tracking Error Sinusoidal Input; A-law		±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$

NOISE(All receive channel measurements are single ended)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted			15	dBrnc0	$VF_XI+ = GRDA$, $VF_XI- = GS_X$
N_{XC2}	Transmit Noise, C-Message Weighted with Eighth Bit Signaling			18	dBrnc0	$VF_XI+ = GRDA$, $VF_XI- = GS_X$; 6th frame signaling
N_{XP}	Transmit Noise, Psophometrically Weighted			−75	dBm0p	$VF_XI+ = GRDA$, $VF_XI- = GS_X$
N_{RC1}	Receive Noise, C-Message Weighted: Quiet Code			11	dBrnc0	$D_R = 11111111$
N_{RC2}	Receive Noise, C-Message Weighted: Sign bit toggle			12	dBrnc0	Input to D_R is zero code with sign bit toggle at 1 KHz rate
N_{RP}	Receive Noise, Psophometrically Weighted			−79	dBm0p	D_R = lowest positive decode level
N_{SF}	Single Frequency Noise End to End Measurement			−50	dBm0	CCITT G.712.4.2 Measure at PWRO+
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit Channel		−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_2$	V_{BB} Power Supply Rejection, Transmit Channel		−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_3$	V_{CC} Power Supply Rejection, Receive Channel		−25		dB	Idle channel; 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz
$PSRR_4$	V_{BB} Power Supply Rejection, Receive Channel		−25		dB	Idle channel; 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz
CT_{TR}	Crosstalk, Transmit to Receive			−71	dB	$VF_XI+ = 0$ dBm0, 1.02 KHz, D_R = lowest positive decode level, measure at PWRO+
CT_{RT}	Crosstalk, Receive to Transmit			−71	dB	$D_R = 0$ dBm0, 1.02 KHz, $VF_XI+ = GRDA$, measure at D_X

DISTORTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
$SD1_X$	Transmit Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_X$	Transmit Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD1_R$	Receive Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_R$	Receive Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2	36 30 25			dB dB dB	0 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X	Transmit Single Frequency Distortion Products			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
DP_R	Receive Single Frequency Distortion Products			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
IMD_1	Intermodulation Distortion, End to End Measurement			−35	dB	CCITT G.712 (7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−49	dBm0	CCITT G.712 (7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−25	dBm0	CCITT G.712 (6.1)
SIS	Spurious in Band Signals, End to End Measurement			−40	dBm0	CCITT G.712 (9)
D_{AX}	Transmit Absolute Delay		245		μs	Fixed Data Rate, CLK_X = 2.048 MHz; 0 dBm0, 1.02 KHz signal at VF_XI+ Measure at D_X.
D_{DX}	Transmit Differential Envelope Delay Relative to D_{AX}		170 95 45 105		μs μs μs μs	f = 500 − 600 Hz f = 600 − 1000 Hz f = 1000 − 2600 Hz f = 2600 − 2800 Hz
D_{AR}	Receive Absolute Delay		190		μs	Fixed Data Rate, CLK_R = 2.048 MHz; Digital input is DMW codes. Measure at PWRO+.
D_{DR}	Receive Differential Envelope Delay Relative to D_{AR}		45 35 85 110		μs μs μs μs	f = 500 − 600 Hz f = 600 − 1000 Hz f = 1000 − 2600 Hz f = 2600 − 2800 Hz

TRANSMIT CHANNEL TRANSFER CHARACTERISTICS

Input amplifier is set for unity gain, noninverting; maximum gain output.

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal input at VF_XI+
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−23	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	

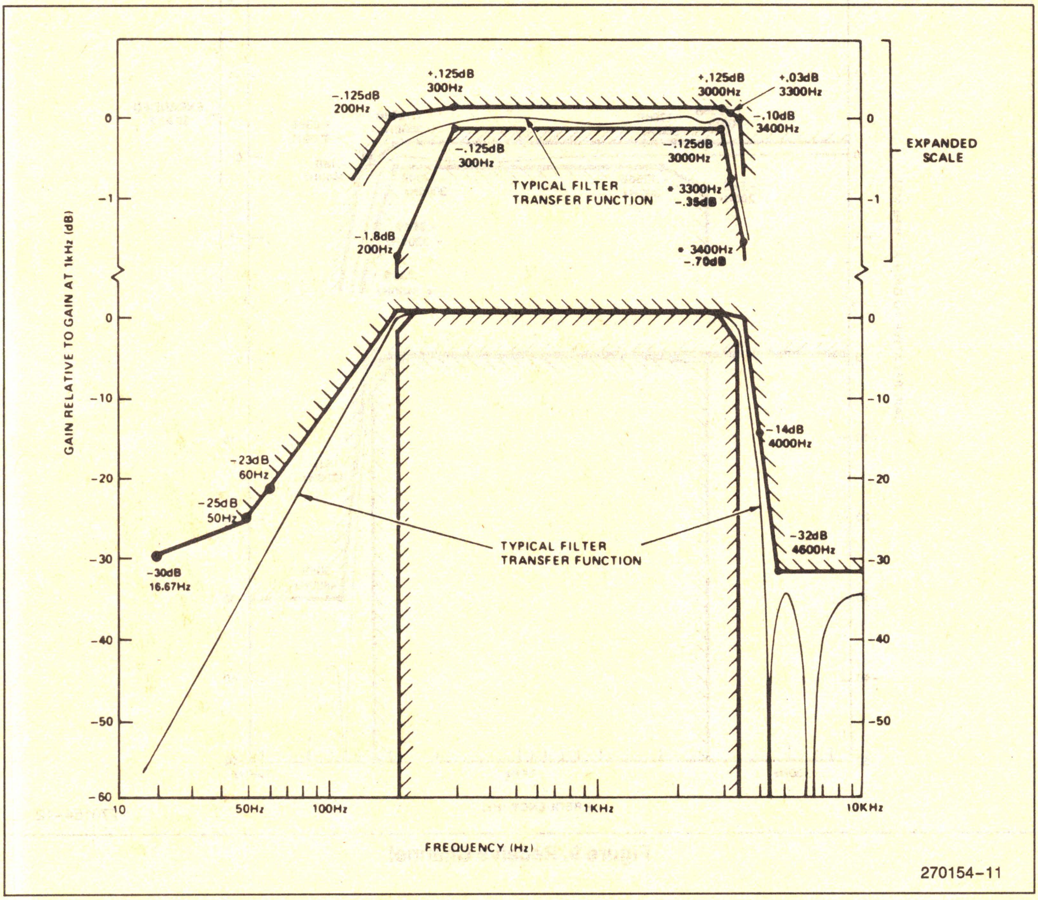


Figure 8. Transmit Channel

RECEIVE CHANNEL TRANSFER CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal input at D_R
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−30	dB	

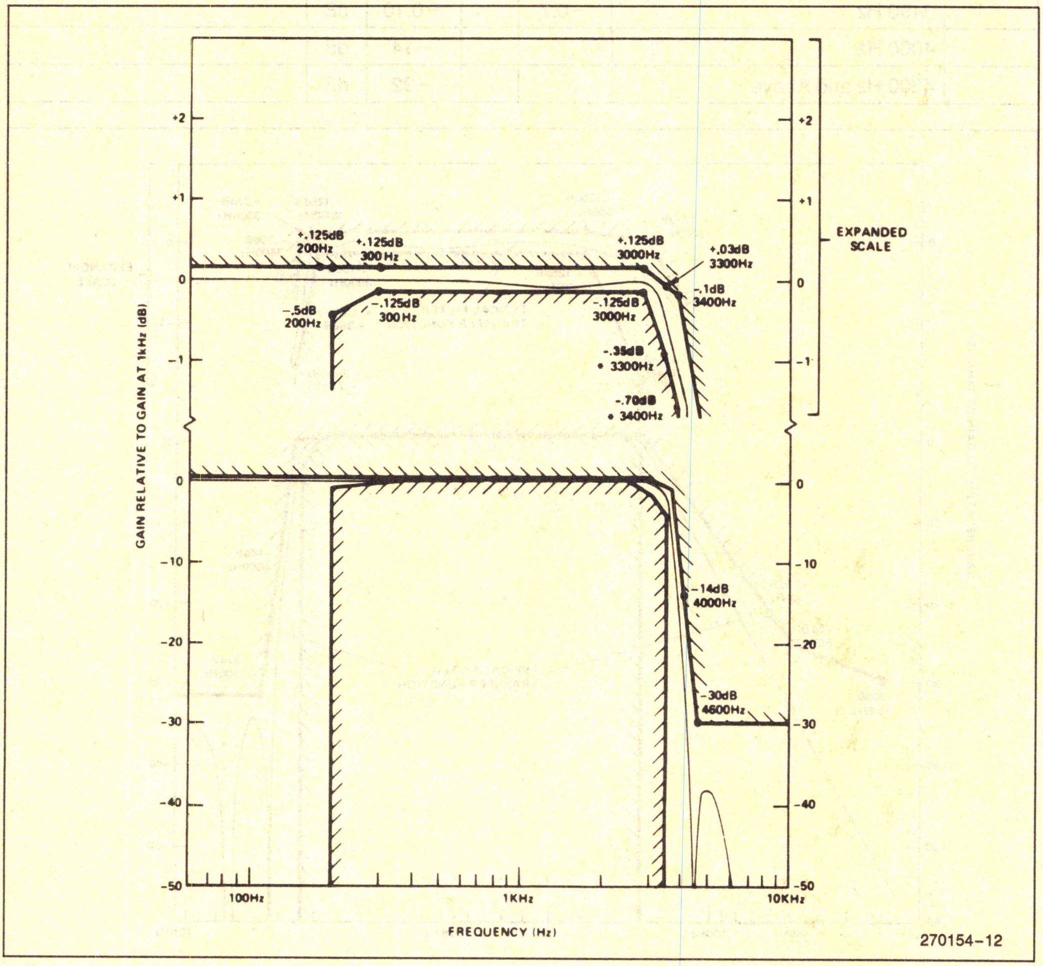


Figure 9. Receive Channel

A.C. CHARACTERISTICS—TIMING PARAMETERS

CLOCK SECTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{CY}	Clock Period, CLK_X, CLK_R	488			ns	$f_{CLKX} = f_{CLKR} = 2.048$ MHz
t_{CLK}	Clock Pulse Width, CLK_X, CLK_R	220			ns	
t_{DCLK}	Data Clock Pulse Width	220			ns	64 KHz $\leq f_{DCLK} \leq$ 2.048 MHz
t_{CDC}	Clock Duty Cycle, CLK_X, CLK_R	45	50	55	%	
t_r, t_f	Clock Rise and Fall Time	5		30	ns	

TRANSMIT SECTION, FIXED DATA RATE MODE(1)

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DZX}	Data Enabled on TS Entry	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{DDX}	Data Delay from CLK_X	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	60		215	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{SOFF}	Timeslot X is Disable	60		145	ns	$0 < C_{LOAD}$
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	
t_{SS}	Signal Setup Time	0			ns	
t_{SH}	Signal Hold Time	0			ns	

RECEIVE SECTION, FIXED DATA RATE MODE

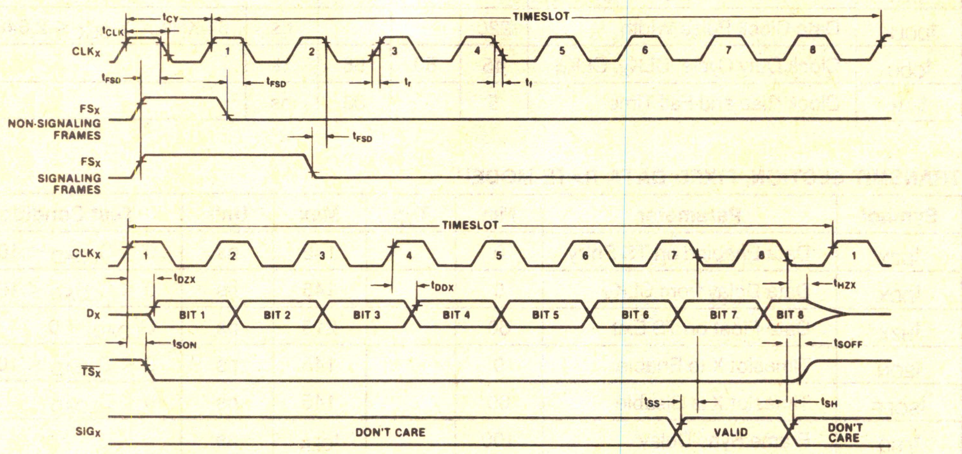
Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DSR}	Receive Data Setup	10			ns	
t_{DHR}	Receive Data Hold	60			ns	
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	
t_{SIGR}	SIG_R Update	0		2	μs	

NOTE:
1. Timing parameters t_{DZX}, t_{HZX}, and t_{SOFF} are referenced to a high impedance state.

WAVEFORMS

Fixed Data Rate Timing

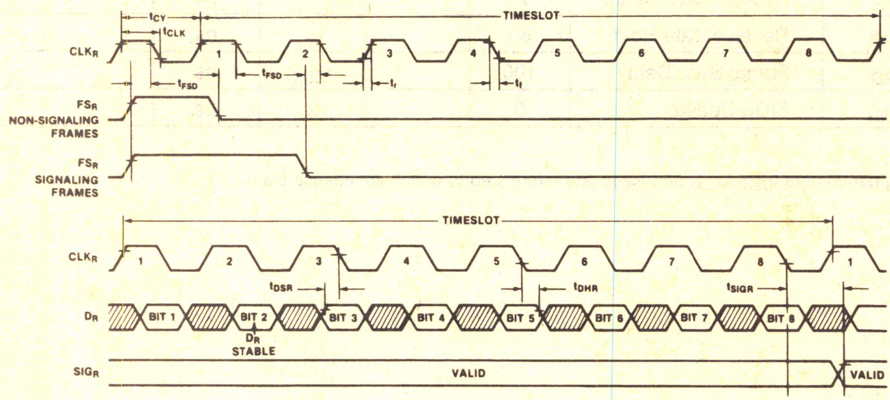
TRANSMIT TIMING



270154-13

NOTE:
All timing parameters referenced to V_{IH} and V_{IL} except t_{DZX}, t_{SOFF} and t_{HZX} which reference a high impedance state.

RECEIVE TIMING



270154-14

NOTE:
All timing parameters referenced to V_{IH} and V_{IL}.

TRANSMIT SECTION, VARIABLE DATA RATE MODE[1]

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDX}	Timeslot Delay from $DCLK_X$[2]	140		$t_{DX} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DDX}	Data Delay from $DCLK_X$	0		100	ns	$0 < C_{LOAD} < 100$ pF
t_{DON}	Timeslot to D_X Active	0		50	ns	$0 < C_{LOAD} < 100$ pF
t_{DOFF}	Timeslot to D_X Inactive	0		80	ns	$0 < C_{LOAD} < 100$ pF
t_{DX}	Data Clock Period	488		15620	ns	
t_{DFSX}	Data Delay from FS_X	0		140	ns	

RECEIVE SECTION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDR}	Timeslot Delay from $DCLK_R$[3]	140		$t_{DR} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DSR}	Data Setup Time	10			ns	
t_{DHR}	Data Hold Time	60			ns	
t_{DR}	Data Clock Period	488		15620	ns	
t_{SER}	Timeslot End Receive Time	0			ns	

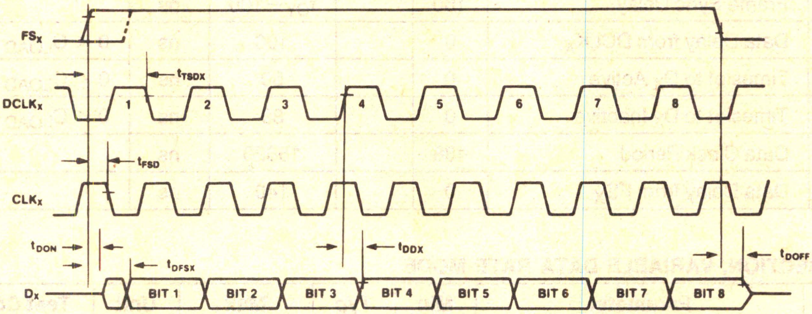
64 KB OPERATION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{FSLX}	Transmit Frame Sync Minimum Downtime	488			ns	FS_X is TTL high for remainder of frame
t_{FSLR}	Receive Frame Sync Minimum Downtime	1952			ns	FS_R is TTL high for remainder of frame
t_{DCLK}	Data Clock Pulse Width			10	μs	

NOTES:
1. Timing parameters t_{DON} and t_{DOFF} are referenced to a high impedance state.
2. t_{FSLX} minimum requirements overrides t_{TSDX} maximum spec for 64 KHz operation.
3. t_{FSLR} minimum requirements overrides t_{TSDR} maximum spec for 64 KHz operation.

VARIABLE DATA RATE TIMING

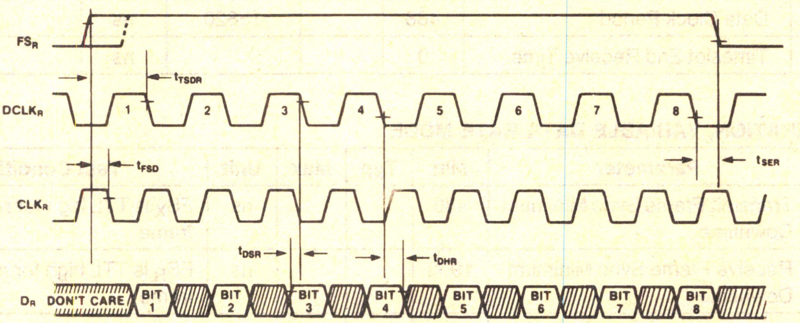
TRANSMIT TIMING



NOTE: 270154–15

All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

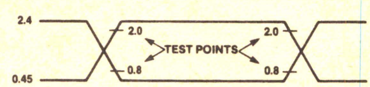
RECEIVE TIMING



NOTE: 270154–16

All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM



270154–17

A.C. Testing: Inputs Are Driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing Measurements Are Made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

2916/2917
HMOS COMBINED SINGLE CHIP PCM CODEC AND FILTER

- **2916 μ-Law, 2.048 MHz Master Clock**
- **2917 A-Law, 2.048 MHz Master Clock**
- **New 16-Pin Package for Higher Linecard Density**
- **AT&T D3/D4 and CCITT Compatible**
- **Variable Timing Mode for Flexible Digital Interface: Supports Data Rates from 64 KB to 2.048 MB**
- **Fully Differential Internal Architecture Enhances Noise Immunity**
- **Fixed Timing Mode for Standard 32-Channel Systems: 2.048 MHz Master Clock**
- **Low Power HMOS-E Technology**
  — 5 mW Typical Power Down
  — 140 mW Typical Operating
- **On Chip Auto Zero, Sample and Hold, and Precision Voltage References**
- **Compatible with Direct Mode Intel 2910A, 2911A, and 2912A Designs**

The Intel 2916 and 2917 are limited feature versions of Intel's 2913 and 2914 combination codec/filter chips. They are fully integrated PCM codecs with transmit/receive filters fabricated in a highly reliable and proven N-channel HMOS silicon gate technology (HMOS-E). These devices provide the functions that were formerly provided by two complex chips (2910A or 2911A and 2912A). Besides the higher level of integration, the performance of the 2916 and 2917 is superior to that of the separate devices.

The primary applications for the 2916 and 2917 are in telephone systems:

- Switching—Digital PBX's and Central Office Switching Systems
- Subscriber Instruments—Digital Handsets and Office Workstations

Other possible applications can be found where the wide dynamic range (78 dB) and minimum conversion time (125 μs) are required for analog to digital interface functions:

- High Speed Modems
- Voice Store and Forward
- Secure Communications
- Digital Echo Cancellation

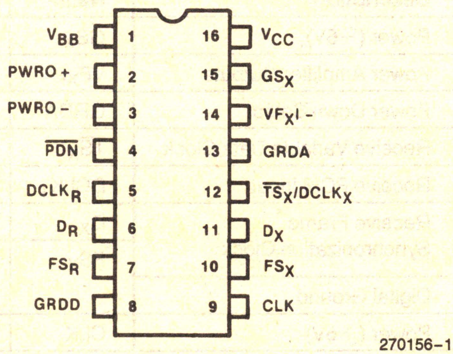


Figure 1. Pin Configuration

November 1986
Order Number: 270156-001

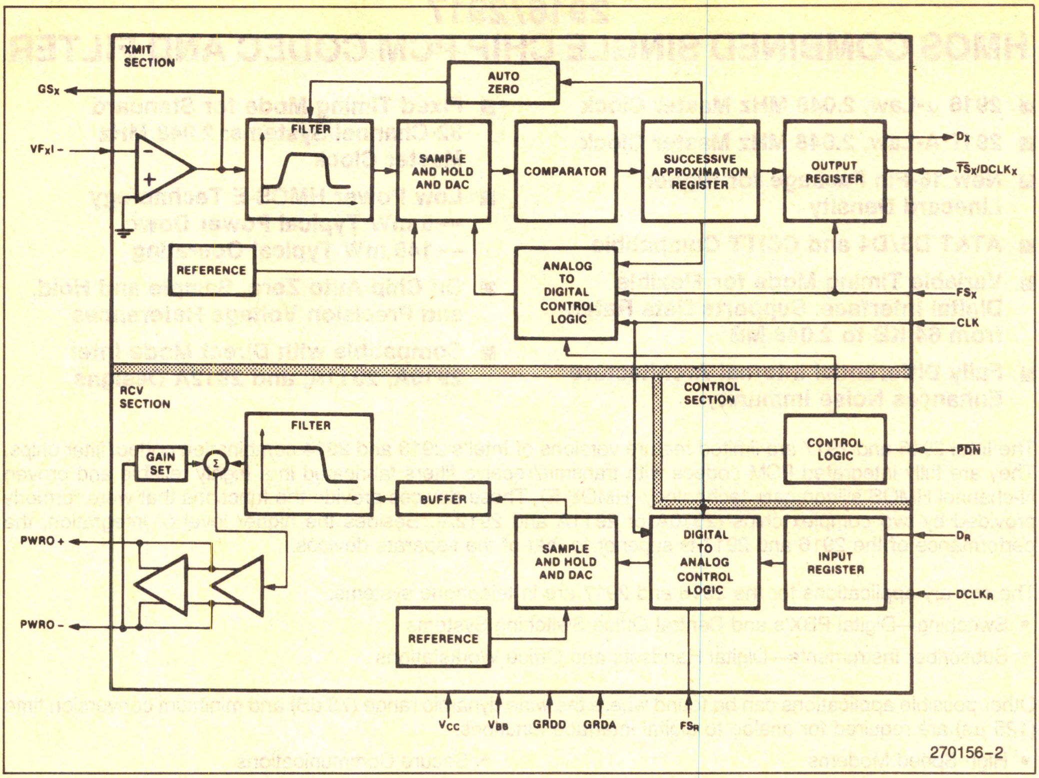


Figure 2. Block Diagram

Table 1. Pin Names

Name	Description	Name	Description
V_{BB}	Power (−5V)	GS_X	Transmit Gain Control
PWRO+, PWRO−	Power Amplifier Outputs	VF_XI-	Analog Input
$\overline{PDN}$	Power Down Select	GRDA	Analog Ground
$DCLK_R$	Receive Variable Data Clock	$\overline{TS}_X$	Timeslot Strobe/Buffer Enable
D_R	Receive PCM Input	$DCLK_X$	Transmit Variable Data Clock
FS_R	Receive Frame Synchronization Clock	D_X	Transmit PCM Output
GRDD	Digital Ground	FS_X	Transmit Frame Synchronization Clock
V_{CC}	Power (+5V)	CLK	Master Clock

Table 2. Pin Description

Symbol	Function
V_{BB}	Most negative supply, input voltage is −5 volts ±5%.
PWRO+	Non-inverting output of power amplifier. Can drive transformer hybrids or high impedance loads directly in either a differential or single ended configuration.
PWRO−	Inverting output of power amplifier. Functionally identical and complementary to PWRO+.
$\overline{PDN}$	Power down select. When $\overline{PDN}$ is TTL high, the device is active. When low, the device is powered down.
$DCLK_R$	Selects the fixed or variable data rate mode. When $DCLK_R$ is connected to V_{BB}, the fixed data rate mode is selected. In this mode, the device is fully compatible with Intel 2910A and 2911A direct mode timing. When $DCLK_R$ is not connected to V_{BB}, the device operates in the variable data rate mode. In this mode $DCLK_R$ becomes the receive data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
D_R	Receive PCM input. PCM data is clocked in on this lead on eight consecutive negative transitions of the receive data clock; CLK in the fixed data rate mode and $DCLK_R$ in variable data rate mode.
FS_R	8 KHz frame synchronization clock input/timeslot enable, receive channel. In variable data rate mode this signal must remain high for the entire length of the timeslot. The receive channel enters the standby state whenever FS_R is TTL low for 300 milliseconds.
GRDD	Digital ground for all internal logic circuits. Not internally tied to GRDA.
CLK	Master and data clock for the fixed data rate mode; master clock only in variable data rate mode.
FS_X	8 KHz frame synchronization clock input/timeslot enable, transmit channel. Operates independently but in an analogous manner to FS_R. The transmit channel enters the standby state whenever FS_X is TTL low for 300 milliseconds.
D_X	Transmit PCM output. PCM data is clocked out on this lead on eight consecutive positive transitions of the transmit data clock; CLK in fixed data rate mode and $DCLK_X$ in variable data rate mode.
$\overline{TS}_X/DCLK_X$	Transmit channel timeslot strobe (output) or data clock (input) for the transmit channel. In fixed data rate mode, this pin is an open drain output designed to be used as an enable signal for a three-state buffer as in 2910A and 2911A direct mode timing. In variable data rate mode, this pin becomes the transmit data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
GRDA	Analog ground return for all internal voice circuits. Not internally connected to GRDD.
VF_XI-	Inverting analog input to uncommitted transmit operational amplifier.
GS_X	Output terminal of on-chip transmit channel input op amp. Internally, this is the voice signal input to the transmit filter.
V_{CC}	Most positive supply; input voltage is +5 volts ±5%.

FUNCTIONAL DESCRIPTION

The 2916 and 2917 provide the analog-to-digital and the digital-to-analog conversions and the transmit and receive filtering necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. They are intended to be used at the analog termination of a PCM line.

The following major functions are provided:

- Bandpass filtering of the analog signals prior to encoding and after decoding
- Encoding and decoding of voice and call progress information
- Encoding and decoding of the signaling and supervision information

GENERAL OPERATION

System Reliability Features

The combochip can be powered up by pulsing FS_X and/or FS_R while a TTL high voltage is applied to $\overline{PDN}$, provided that all clocks and supplies are connected. The 2916 and 2917 have internal resets on power up (or when V_{BB} or V_{CC} are re-applied) in order to ensure validity of the digital outputs and thereby maintain integrity of the PCM highway.

On the transmit channel, digital outputs D_X and $\overline{TS}_X$ are held in a high impedance state for approximately four frames (500 μs) after power up or application of V_{BB} or V_{CC}. After this delay, D_X and $\overline{TS}_X$ will be functional and will occur in the proper timeslot. The analog circuits on the transmit side require approximately 60 milliseconds to reach their equilibrium value due to the autozero circuit settling time.

To enhance system reliability, $\overline{TS}_X$ and D_X will be placed in a high impedance state approximately 30 μs after an interruption of CLK.

Power Down and Standby Modes

To minimize power consumption, two power down modes are provided in which most 2916/2917 functions are disabled. Only the power down, clock, and frame sync buffers, which are required to power up the device, are enabled in these modes. As shown in Table 3, the digital outputs on the appropriate channels are placed in a high impedance state until the device returns to the active mode.

The Power Down mode utilizes an external control signal to the $\overline{PDN}$ pin. In this mode, power consumption is reduced to an average of 5 mW. The device is active when the signal is high and inactive when it is low. In the absence of any signal, the $\overline{PDN}$ pin floats to TTL high allowing the device to remain active continuously.

The Standby mode leaves the user an option of powering either channel down separately or powering the entire device down by selectively removing FS_X and/or FS_R. With both channels in the standby state, power consumption is reduced to an average of 12 mW. If transmit only operation is desired, FS_X should be applied to the device while FS_R is held low. Similarly, if receive only operation is desired, FS_R should be applied while FS_X is held low.

Fixed Data Rate Mode

Fixed data rate timing, which is 2910A and 2911A compatible, is selected by connecting $DCLK_R$ to V_{BB}. It employs master clock CLK, frame synchronization clocks FS_X and FS_R, and output $\overline{TS}_X$.

Table 3. Power-Down Methods

Device Status	Power-Down Method	Typical Power Consumption	Digital Output Status
Power Down Mode	$\overline{PDN}$ = TTL Low	5 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 10 μs.
Standby Mode	FS_X and FS_R are TTL Low	12 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 300 milliseconds.
Only Transmit is on Standby	FS_X is TTL Low	70 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 300 milliseconds.
Only Receive is on Standby	FS_R is TTL Low	110 mW	

CLK serves as the master clock to operate the codec and filter sections and as the bit clock to clock the data in and out from the PCM highway. FS_X and FS_R are 8 KHz inputs which set the sampling frequency. $\overline{TS}_X$ is a timeslot strobe/buffer enable output which gates the PCM word onto the PCM highway when an external buffer is used to drive the line.

Data is transmitted on the highway at D_X on the first eight positive transitions of CLK following the rising edge of FS_X. Similarly, on the receive side, data is received on the first eight falling edges of CLK. The frequency of CLK must be 2.048 MHz. No other frequency of operation is allowed in the fixed data rate mode.

Variable Data Rate Mode

Variable data rate timing is selected by connecting $DCLK_R$ to the bit clock for the receive PCM highway rather than to V_{BB}. It employs master clock CLK, bit clocks $DCLK_R$ and $DCLK_X$, and frame synchronization clocks FS_R and FS_X.

Variable data rate timing allows for a flexible data frequency. It provides the ability to vary the frequency of the bit clocks, from 64 KHz to 2.048 MHz. The master clock is still restricted to 2.048 MHz.

In this mode, $DCLK_R$ and $DCLK_X$ become the data clocks for the receive and transmit PCM highways. While FS_X is high, PCM data from D_X is transmitted onto the highway on the next eight consecutive positive transitions of $DCLK_X$. Similarly, while FS_R is high, each PCM bit from the highway is received by D_R on the next eight consecutive negative transitions of $DCLK_R$.

On the transmit side, the PCM word will be repeated in all remaining timeslots in the 125 μs frame as long as $DCLK_X$ is pulsed and FS_X is held high. This feature allows the PCM word to be transmitted to the PCM highway more than once per frame, if desired, and is only available in the variable data rate mode.

Precision Voltage References

No external components are required with the combochip to provide the voltage reference function. Voltage references are generated on-chip and are calibrated during the manufacturing process. The technique uses a difference in sub-surface charge density between two suitably implanted MOS devices to derive a temperature and bias stable reference voltage. These references determine the gain and dynamic range characteristics of the device.

Separate references are supplied to the transmit and receive sections and each is trimmed independently during the manufacturing process. The reference value is then further trimmed in the gain setting op-amps to a final precision value. With this method the combochip can achieve the extremely accurate Digital Milliwatt Responses specified in the TRANSMISSION PARAMETERS, providing the user a significant margin for error in other board components.

TRANSMIT OPERATION

Transmit Filter

The input section provides gain adjustment in the passband by means of an on-chip operational amplifier. This operational amplifier has a common mode range of ±2.17 volts, a maximum DC offset of 25 mV, a minimum open loop voltage gain of 5000, and a unity gain bandwidth of typically 1 MHz. Gain of up to 20 dB can be set without degrading the performance of the filter. The load impedance to ground (GRDA) at the amplifier output (GS_X) must be greater than 10 kilohms in parallel with less than 50 pF. The input signal on lead VF_XI- can be either AC or DC coupled. The input op amp can only be used in the inverting mode as shown in Figure 3.

A low pass anti-aliasing section is included on-chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

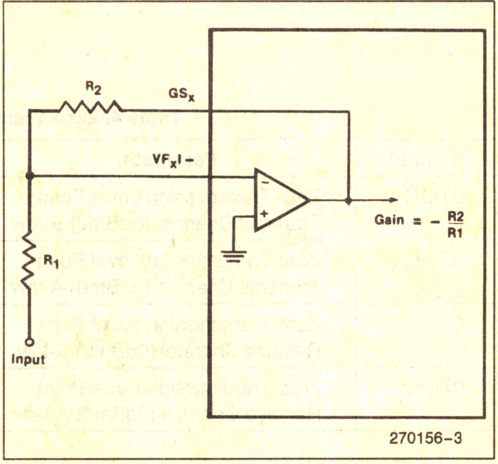


Figure 3. Transmit Filter Gain Adjustment

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 channel bank transmission specification and CCITT recommendation G.712. The 2916 and 2917 specifications meet or exceed digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 4.

A high pass section configuration was chosen to reject low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Even though there is high rejection at these frequencies, the sharpness of the band edge gives low attenuation at 200 Hz. This feature allows the use of low-cost transformer hybrids without external components.

Encoding

The encoder internally samples the output of the transmit filter and holds each sample on an internal sample and hold capacitor. The encoder then performs an analog to digital conversion on a switched capacitor array. Digital data representing the sample is transmitted on the first eight data clock bits of the next frame.

An on-chip autozero circuit corrects for DC-offset on the input signal to the encoder. This autozero circuit uses the sign bit averaging technique; the sign bit from the encoder output is long term averaged and subtracted from the input to the encoder. In this way, all DC offset is removed from the encoder input waveform.

RECEIVE OPERATION

Decoding

The PCM word at the D_R lead is serially fetched on the first eight data clock bits of the frame. A D/A conversion is performed on the digital word and the corresponding analog sample is held on an internal sample and hold capacitor. This sample is then transferred to the receive filter.

Receive Filter

The receive filter provides passband flatness and stopband rejection which fulfills both the AT&T D3/D4 specification and CCITT recommendation G.712. The filter contains the required compensation for the (sin x)/x response of such decoders. The receive filter characteristics and specifications will be within the limits shown in Figure 5.

Receive Output Power Amplifiers

A balanced output amplifier is provided in order to allow maximum flexibility in output configuration. Either of the two outputs can be used single ended (referenced to GRDA) to drive single ended loads. Alternatively, the differential output will drive a bridged load directly. The output stage is capable of driving loads as low as 300 ohms single ended or 600 ohms differentially.

Transmission levels are specified relative to the receive channel output under digital milliwatt conditions, that is, when the digital input at D_R is the eight-code sequence specified in CCITT recommendation G.711.

Table 4. Zero Transmission Level Points

Symbol	Parameter	Value	Units	Test Conditions
$0TLP1_X$	Zero Transmission Level Point Transmit Channel (0dBm0) μ-law	+2.76 +1.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_X$	Zero Transmission Level Point Transmit Channel (0dBm0) A-law	+2.79 +1.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP1_R$	Zero Transmission Level Point Receive Channel (0dBm0) μ-law	+5.76 +4.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_R$	Zero Transmission Level Point Receive Channel (0dBm0) A-law	+5.79 +4.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias .... −10°C to +80°C
Storage Temperature .... −65°C to +150°C
V_{CC} and GRDD with Respect to V_{BB} .... −0.3V to +15V
All Input and Output Voltages with Respect to V_{BB} .... −0.3V to +15V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

(T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified)

Typical values are for T_A = 25°C and nominal power supply values.

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{IL}	Low Level Input Current			10	μA	GRDD ≤ V_{IN} ≤ V_{IL} (Note 1)
I_{IH}	High Level Input Current			10	μA	V_{IH} ≤ V_{IN} ≤ V_{CC}
V_{IL}	Input Low Voltage			0.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 3.2 mA at D_X, $\overline{TS}_X$
V_{OH}	Output High Voltage	2.4			V	I_{OH} = 9.6 mA at D_X
C_{OX}	Digital Output Capacitance(2)		5		pF	
C_{IN}	Digital Input Capacitance		5	10	pF	

POWER DISSIPATION

All measurements made at f_{DCLK} = 2.048 MHz, outputs unloaded.

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{CC1}	V_{CC} Operating Current(4)		14	19	mA	
I_{BB1}	V_{BB} Operating Current		−18	−24	mA	
I_{CC0}	V_{CC} Power Down Current		0.5	1.0	mA	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
I_{BB0}	V_{BB} Power Down Current		−0.5	−1.0	mA	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
I_{CCS}	V_{CC} Standby Current		1.2	2.4	mA	FS_X, FS_R ≤ V_{IL}; after 300 ms
I_{BBS}	V_{BB} Standby Current		−1.2	−2.4	mA	FS_X, FS_R ≤ V_{IL}; after 300 ms
P_{D1}	Operating Power Dissipation(3)		140	200	mW	
P_{D0}	Power Down Dissipation(3)		5	10	mW	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
P_{ST}	Standby Power Dissipation(3)		12	25	mW	FS_X, FS_R ≤ V_{IL}

NOTES:
1. V_{IN} is the voltage on any digital pin.
2. Timing parameters are guaranteed based on a 100 pF load capacitance. Up to eight digital outputs may be connected to a common PCM highway without buffering, assuming a board capacitance of 60 pF.
3. With nominal power supply values.
4. V_{CC} applied last or simultaneously with V_{BB}.

ANALOG INTERFACE, TRANSMIT CHANNEL INPUT STAGE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
I_{BX1}	Input Leakage Current, VF_XI-			100	nA	$-2.17V \le V_{IN} \le 2.17V$
R_{IXI}	Input Resistance, VF_XI-	10			MΩ	
V_{OSXI}	Input Offset Voltage, VF_XI-			25	mV	
A_{VOL}	DC Open Loop Voltage Gain, GS_X	5000				
f_C	Open Loop Unity Gain Bandwidth, GS_X		1		MHz	
C_{LXI}	Load Capacitance, GS_X			50	pF	
R_{LXI}	Minimum Load Resistance, GS_X	10			KΩ	

ANALOG INTERFACE, RECEIVE CHANNEL DRIVER AMPLIFIER STATE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
R_{ORA}	Output Resistance, PWRO+, PWRO−		1		Ω	
V_{OSRA}	Single-Ended Output DC Offset, PWRO+, PWRO−		75		mV	Relative to GRDA
C_{LRA}	Load Capacitance, PWRO+, PWRO−			100	pF	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

Unless otherwise noted, the analog input is a 0 dBm0, 1020 Hz sine wave.(1) Input amplifier is set for unity gain, inverting. The digital input is a PCM bit stream generated by passing a 0 dBm0, 1020 Hz sine wave through an ideal encoder. Receive output is measured single ended. All output levels are (sin x)/x corrected. Typical values are for $T_A = 25°C$ and nominal power supply values. ($T_A = 0°C$ to $+70°C$; $V_{CC} = +5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0V; GRDD = 0V; unless otherwise specified).

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response (Transmit Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Signal Input of 1.064 Vrms μ-law Signal Input of 1.068 Vrms A-law $T_A = 25°C$, $V_{BB} = -5V$, $V_{CC} = +5V$
EmW_{TS}	EmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% Supplies, 0 to 70°C Relative to Nominal Conditions
DmW	Digital Milliwatt Response (Receive Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Measure Relative to $0TLP_R$. Signal Input per CCITT Recommendation G.711. Output Signal of 1000 Hz. $R_L = \infty$ $T_A = 25°C$; $V_{BB} = -5V$, $V_{CC} = +5V$.
DmW_{TS}	DmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% Supplies, 0 to 70°C

NOTE:

1. 0 dBm0 is defined as the zero reference point of the channel under test (0TLP). This corresponds to an analog signal input of 1.064 volts rms or an output of 1.503 volts rms (for μlaw).

GAIN TRACKING

Reference Level = −10 dBm0

Symbol	Parameter	2916 Min	2916 Max	2917 Min	2917 Max	Unit	Test Conditions
$GT1_X$	Transmit Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2			dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT2_X$	Transmit Gain Tracking Error Sinusoidal Input; A-law				±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT1_R$	Receive Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2			dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$
$GT2_R$	Receive Gain Tracking Error Sinusoidal Input; A-law				±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, $R_L = 300\Omega$

NOISE (All receive channel measurements are single ended)

Symbol	Parameter	2916 Min	2916 Typ	2916 Max	2917 Min	2917 Typ	2917 Max	Unit	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted			15				dBrncO	Unity Gain
N_{XP}	Transmit Noise, Psophometrically Weighted						−75	dBm0p	Unity Gain
N_{RC1}	Receive Noise, C-Message Weighted: Quiet Code			11				dBrncO	D_R = 11111111
N_{RC2}	Receive Noise, C-Message Weighted: Sign Bit Toggle			12				dBrncO	Input to D_R is Zero Code with Sign Bit Toggle at 1 KHz Rate
N_{RP}	Receive Noise, Psophometrically Weighted						−79	dBm0p	D_R = Lowest Positive Decode Level
N_{SF}	Single Frequency Noise End to End Measurement			−50			−50	dBm0	CCITT G.712.4.2 Measure at PWRO+
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle Channel; 200 mV P-P Signal on Supply; 0 to 50 KHz, Measure at D_X
$PSRR_2$	V_{BB} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle Channel; 200 mV P-P Signal on Supply; 0 to 50 KHz, Measure at D_X
$PSRR_3$	V_{CC} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle Channel; 200 mV P-P Signal on Supply; Measure Narrow Band at PWRO+, 0 to 50 KHz
$PSRR_4$	V_{BB} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle Channel; 200 mV P-P Signal on Supply; Measure Narrow Band at PWRO+, 0 to 50 KHz
CT_{TR}	Crosstalk, Transmit to Receive			−71			−71	dB	Input = 0 dBm0, Unity Gain, 1.02 KHz, D_R = Lowest Positive Decode Level, Measure at PWRO+
CT_{RT}	Crosstalk, Receive to Transmit			−71			−71	dB	D_R = 0 dBm0, 1.02 KHz, Measure at D_X

DISTORTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
$SD1_X$	Transmit Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2 (2916)	36 30 25			dB dB dB	0 dBm0 to −30 dBm0 −30 dBm0 to −40 dBm0 −40 dBm0 to −45 dBm0
$SD2_X$	Transmit Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2 (2917)	36 30 25			dB dB dB	0 dBm0 to −30 dBm0 −30 dBm0 to −40 dBm0 −40 dBm0 to −45 dBm0
$SD1_R$	Receive Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2 (2916)	36 30 25			dB dB dB	0 dBm0 to −30 dBm0 −30 dBm0 to −40 dBm0 −40 dBm0 to −45 dBm0
$SD2_R$	Receive Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2 (2917)	36 30 25			dB dB dB	0 dBm0 to −30 dBm0 −30 dBm0 to −40 dBm0 −40 dBm0 to −45 dBm0
DP_X	Transmit Single Frequency Distortion Products (2916)			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
DP_R	Receive Single Frequency Distortion Products (2916)			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
IMD_1	Intermodulation Distortion, End to End Measurement			−35	dB	CCITT G.712 (7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−49	dBm0	CCITT G.712 (7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−25	dBm0	CCITT G.712 (6.1)
SIS	Spurious In Band Signals, End to End Measurement			−40	dBm0	CCITT G.712 (9)
D_{AX}	Transmit Absolute Delay		245		μs	Fixed Data Rate. CLK_X = 2.048 MHz; 0 dBm0, 1.02 KHz Input Signal, Unity Gain. Measure at D_X.
D_{DX}	Transmit Differential Envelope Delay Relative to D_{AX}		170 95 45 105		μs μs μs μs	f = 500 Hz to 600 Hz f = 600 Hz to 1000 Hz f = 1000 Hz to 2600 Hz f = 2600 Hz to 2800 Hz
D_{AR}	Receive Absolute Delay		190		μs	Fixed Data Rate, CLK = 2.048 MHz; Digital Input is DMW Codes. Measure at PWRO+
D_{DR}	Receive Differential Envelope Delay Relative to D_{AR}		45 35 85 110		μs μs μs μs	f = 500 Hz to 600 Hz f = 600 Hz to 1000 Hz f = 1000 Hz to 2600 Hz f = 2600 Hz to 2800 Hz

TRANSMIT CHANNEL TRANSFER CHARACTERISTICS

Input amplifier is set for unity gain, inverting.

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at VF_XI-
	16.67 Hz			−30	dB	
	50 Hz			−25	dB	
	60 Hz			−23	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	

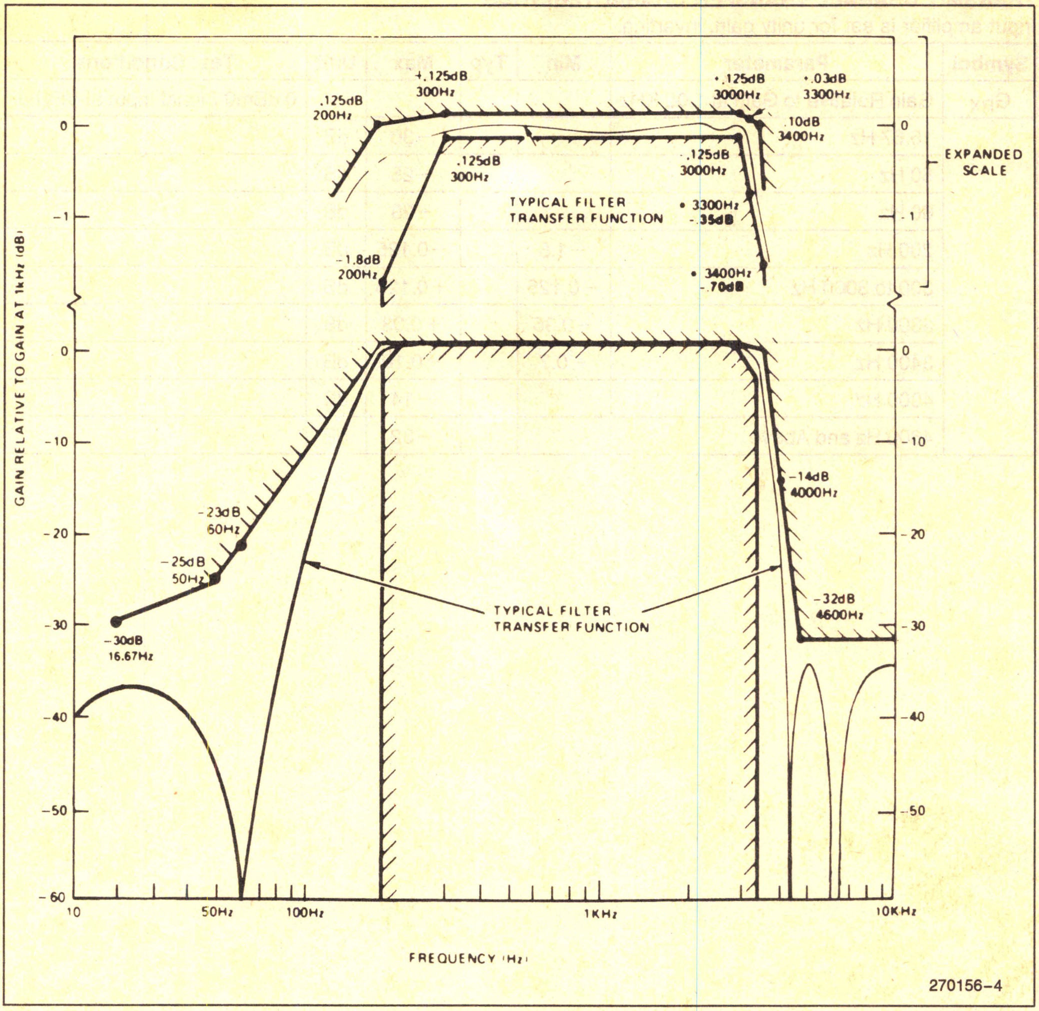


Figure 4. Transmit Channel

RECEIVE CHANNEL TRANSFER CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at D_R
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	+0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−30	dB	

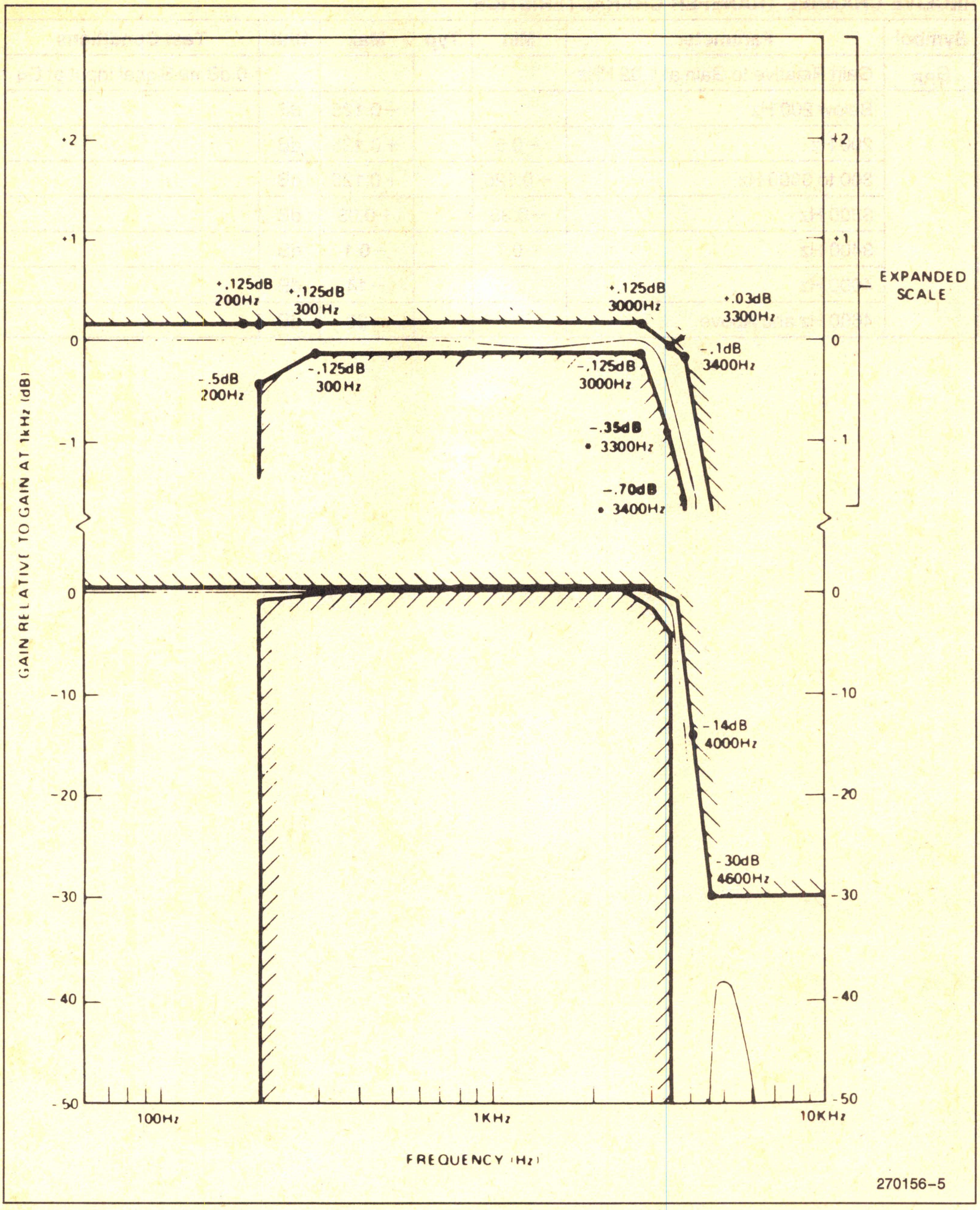


Figure 5. Receive Channel

A.C. CHARACTERISTICS—TIMING PARAMETERS

CLOCK SECTION

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{CY}	Clock Period, CLK	488			ns	$f_{CLK} = 2.048$ MHz
t_{CLK}	Clock Pulse Width, CLK	220			ns	
t_{DCLK}	Data Clock Pulse Width	220			ns	64 KHz $\leq f_{DCLK} \leq$ 2.048 MHz
t_{CDC}	Clock Duty Cycle, CLK	45	50	55	%	
t_r, t_f	Clock Rise and Fall Time	5		30	ns	

TRANSMIT SECTION, FIXED DATA RATE MODE[1]

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DZX}	Data Enabled on TS Entry	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{DDX}	Data Delay from CLK	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	60		215	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{SOFF}	Timeslot X to Disable	60		215	ns	$C_{LOAD} = 0$
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	

RECEIVE SECTION, FIXED DATA RATE MODE

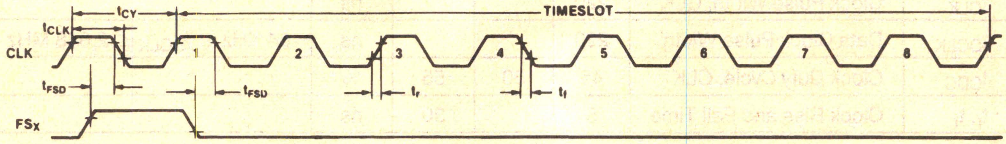
Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{DSR}	Receive Data Setup	10			ns	
t_{DHR}	Receive Data Hold	60			ns	
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	

NOTE:
1. Timing parameters T_{DZX}, T_{HZX}, and T_{SOFF} are referenced to a high impedance state.

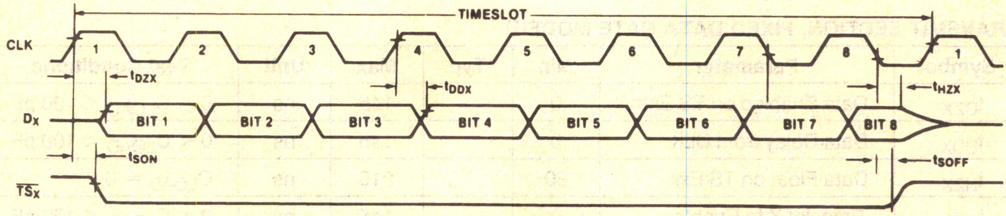
WAVEFORMS

Fixed Data Rate Timing

TRANSMIT TIMING



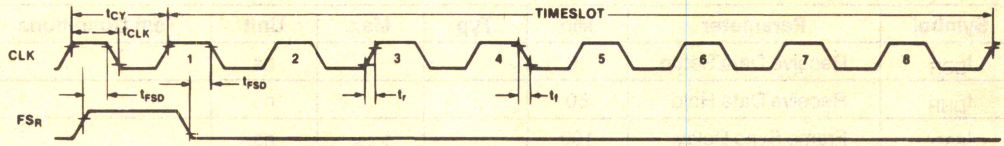
270156-6



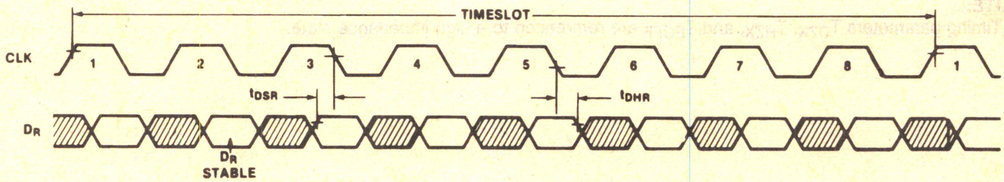
270156-7

NOTE:
1. All timing parameters referenced to V_{IH} and V_{IL} except t_{DZX}, t_{SOFF} and t_{HZX} which reference a high impedance state.

RECEIVE TIMING



270156-8



270156-9

NOTE:
1. All timing parameters referenced to V_{IH} and V_{IL}.

TRANSMIT SECTION, VARIABLE DATA RATE MODE[1]

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDX}	Timeslot Delay from $DCLK_X$[2]	140		$t_{DX}-140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY}-100$	ns	
t_{DDX}	Data Delay from $DCLK_X$	0		100	ns	$0 < C_{LOAD} < 100$ pF
t_{DON}	Timeslot to D_X Active	0		50	ns	$0 < C_{LOAD} < 100$ pF
t_{DOFF}	Timeslot to D_X Inactive	0		80	ns	$0 < C_{LOAD} < 100$ pF
t_{DX}	Data Clock Period	488		15620	ns	
t_{DFSX}	Data Delay from FS_X	0		140	ns	

RECEIVE SECTION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{TSDR}	Timeslot Delay from $DCLK_R$[3]	140		$t_{DR}-140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY}-100$	ns	
t_{DSR}	Data Setup Time	10			ns	
t_{DHR}	Data Hold Time	60			ns	
t_{DR}	Data Clock Period	488		15620	ns	
t_{SER}	Timeslot End Receive Time	60			ns	

64 KB OPERATION, VARIABLE DATA RATE MODE

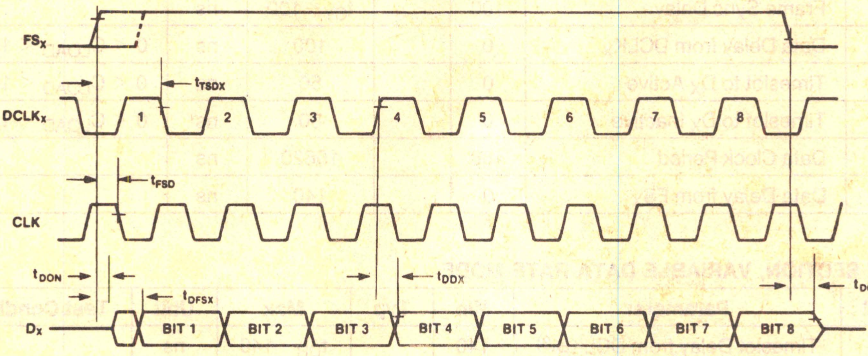
Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
t_{FSLX}	Transmit Frame Sync Minimum Downtime	488			ns	FS_X is TTL High for Remainder of Frame
t_{FSLR}	Receive Frame Sync Minimum Downtime	1952			ns	FS_R is TTL High for Remainder of Frame
t_{DCLK}	Data Clock Pulse Width			10	μs	

NOTES:

1. Timing parameters t_{DON} and t_{DOFF} are referenced to a high impedance state.
2. t_{FSLX} minimum requirements overrides t_{TSDX} maximum spec for 64 KHz operation.
3. t_{FSLR} minimum requirements overrides t_{TSDR} maximum spec for 64 KHz operation.

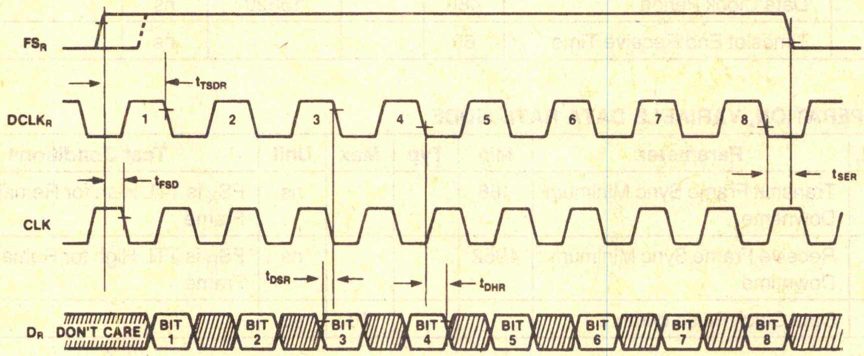
VARIABLE DATA RATE TIMING

TRANSMIT TIMING



270156-10

RECEIVE TIMING

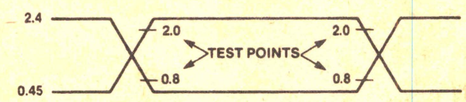


270156-11

NOTE:
1. All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM

INPUT/OUTPUT



270156-12

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

29C16 AND 29C17
16 PIN CHMOS SINGLE CHIP PCM CODEC AND FILTER

- **29C16 μ-Law, 2.048 MHz Master Clock**
- **29C17 A-Law, 2.048 MHz Master Clock**

- **Low-Power Pin Compatible Version of Intel's 2916 and 2917**
- **AT&T D3/D4 and CCITT Compatible**
- **16-Pin Package for Higher Linecard Densities**
- **Ideal for Digital Handset Applications**
- **3 Low-Power Modes**
  - **—5 mW Typical Power Down**
  - **—8 mW Typical Standby**
  - **—70 mW Typical Operating**
- **TTL and CMOS Compatible**
- **Two Timing Modes**
  - **—64 KHz to 2 MHz Variable**
  - **—2 MHz Direct**

Intel's 29C16 and 29C17 are CHMOS versions of Intel's NMOS 2916 and 2917 family members. CHMOS is a technology built on HMOS-II, thus realizing the high performance and density obtained in that process while achieving the low power consumption typical of CMOS circuits.

The 29C16 and 29C17 are limited feature versions of the 29C13 and 29C14. The inherent low-power and small package size make these devices ideal for digital handset and cellular telephones where small size and low power are especially desirable.

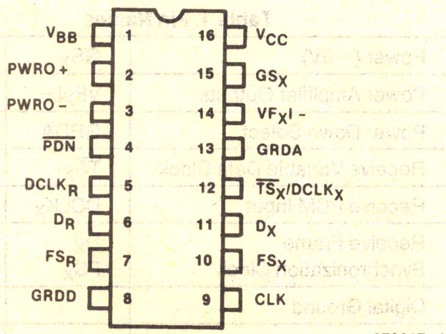


Figure 1. Pin Configuration

November 1986
Order Number: 270217-001

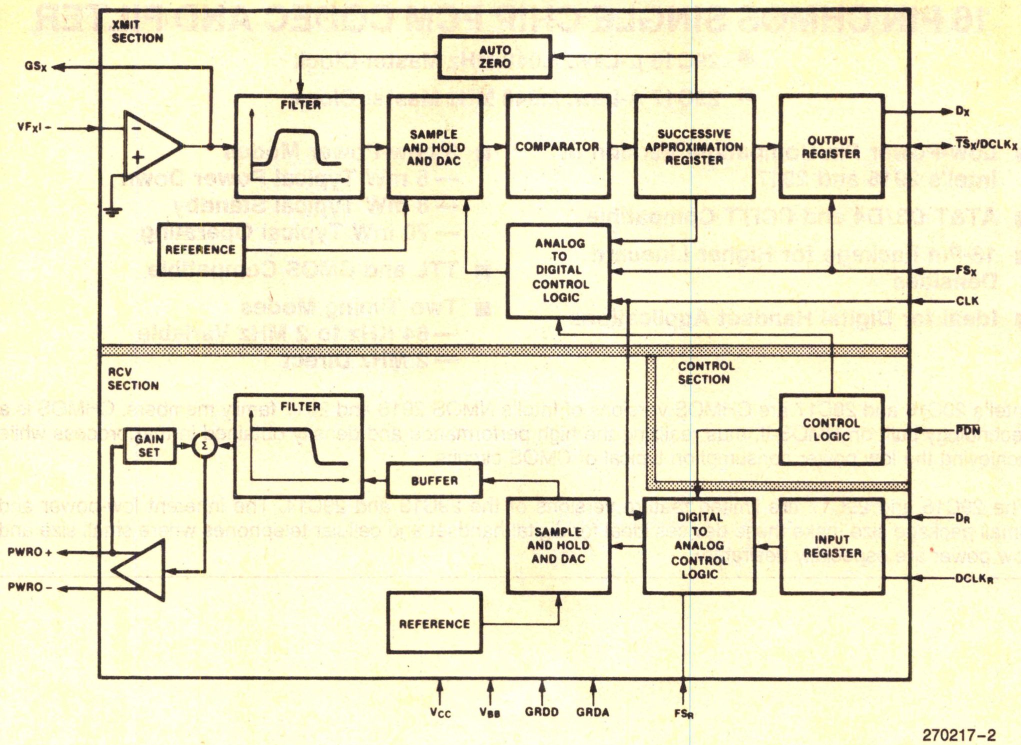


Figure 2. Block Diagram

Table 1. Pin Names

V_{BB}	Power (−5V)	GS_X	Transmit Gain Control
PWRO+, PWRO−	Power Amplifier Outputs	VF_XI-	Analog Input
$\overline{PDN}$	Power Down Select	GRDA	Analog Ground
$DCLK_R$	Receive Variable Data Clock	$\overline{TS}_X$	Timeslot Strobe/Buffer Enable
D_R	Receive PCM Input	$DCLK_X$	Transmit Variable Data Clock
FS_R	Receive Frame Synchronization Clock	D_X FS_X	Transmit PCM Output Transmit Frame
GRDD	Digital Ground		Synchronization Clock
V_{CC}	Power (+5V)	CLK	Master Clock

Table 2. Pin Description

Symbol	Function
V_{BB}	Most negative supply; input voltage is −5V ±5%.
PWRO+	Non-inverting output of power amplifier. Can drive transformer hybrids or high impedance loads directly in either a differential or single ended configuration.
PWRO−	Inverting output of power amplifier. Functionally identical and complementary to PWRO+.
$\overline{PDN}$	Power down select. When $\overline{PDN}$ is TTL high, the device is active. When low, the device is powered down.
$DCLK_R$	Selects the fixed or variable data rate mode. When $DCLK_R$ is connected to V_{BB}, the fixed data rate mode is selected. In this mode, the device is fully compatible with Intel 2910A and 2911A direct mode timing. When $DCLK_R$ is not connected to V_{BB}, the device operates in the variable data rate mode. In this mode $DCLK_R$ becomes the receive data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
D_R	Receive PCM input. PCM data is clocked in on this lead on eight consecutive negative transitions of the receive data clock; CLK in the fixed data rate mode and $DCLK_R$ in variable data rate mode.
FS_R	8 KHz frame synchronization clock input/timeslot enable, receive channel. In variable data rate mode this signal must remain high for the entire length of the timeslot. The receive channel enters the standby state whenever FS_R is TTL low for 300 milliseconds.
GRDD	Digital ground for all internal logic circuits. Not internally tied to GRDA.
CLK	Master and data clock for the fixed data rate mode; master clock only in variable data rate mode.
FS_X	8 KHz frame synchronization clock input/timeslot enable, transmit channel. Operates independently but in an analogous manner to FS_R. The transmit channel enters the standby state whenever FS_X is TTL low for 300 milliseconds.
D_X	Transmit PCM output. PCM data is clocked out on this lead on eight consecutive positive transitions of the transmit data clock; CLK in fixed data rate mode and $DCLK_X$ in variable data rate mode.
$\overline{TS}_X/DCLK_X$	Transmit channel timeslot strobe (output) or data clock (input) for the transmit channel. In fixed data rate mode, this pin is an open drain output designed to be used as an enable signal for a three-state buffer as in 2910A and 2911A direct mode timing. In variable data rate mode, this pin becomes the transmit data clock which operates at TTL levels from 64 Kb to 2.048 Mb data rates.
GRDA	Analog ground return for all internal voice circuits. Not internally connected to GRDD.
VF_XI-	Inverting analog input to uncommitted transmit operational amplifier.
GS_X	Output terminal of on-chip transmit channel input op amp. Internally, this is the voice signal input to the transmit filter.
V_{CC}	Most positive supply; input voltage is +5V ±5%.

FUNCTIONAL DESCRIPTION

The 29C16 and 29C17 provide the analog-to-digital and the digital-to-analog conversions and the transmit and receive filtering necessary to interface a full duplex (4 wires) voice telephone circuit with the PCM highways of a time division multiplexed (TDM) system. They are intended to be used at the analog termination of a PCM line.

The following major functions are provided:

- Bandpass filtering of the analog signals prior to encoding and after decoding
- Encoding and decoding of voice and call progress information
- Encoding and decoding of the signaling and supervision information

GENERAL OPERATION

System Reliability Features

The combochip can be powered up by pulsing FS_X and/or FS_R while a TTL high voltage is applied to $\overline{PDN}$, provided that all clocks and supplies are connected. The 29C16 and 29C17 have internal resets on power up (or when V_{BB} or V_{CC} are re-applied) in order to ensure validity of the digial outputs and thereby maintain integrity of the PCM highway.

On the transmit channel, digital outputs D_X and $\overline{TS}_X$ are held in a high impedance state for approximately four frames (500 μs) after power up or application of V_{BB} or V_{CC}. After this delay, D_X and $\overline{TS}_X$ will be functional and will occur in the proper timeslot. The analog circuits on the transmit side require approximately 60 milliseconds to reach their equilibrium value due to the autozero circuit settling time.

To enhance system reliability, $\overline{TS}_X$ and D_X will be placed in a high impedance state approximately 30 μs after an interruption of CLK.

Power Down and Standby Modes

To minimize power consumption, two power down modes are provided in which most 29C16/C17 functions are disabled. Only the power down, clock, and frame sync buffers, which are required to power up the device, are enabled in these modes. As shown in Table 3, the digital outputs on the appropriate channels are placed in a high impedance state until the device returns to the active mode.

The Power Down mode utilizes an external control signal to the $\overline{PDN}$ pin. In this mode, power consumption is reduced to the value shown in Table 3. The device is active when the signal is high and inactive when it is low. In the absence of any signal, the $\overline{PDN}$ pin floats to TTL high allowing the device to remain active continuously.

The Standby mode leaves the user an option of powering either channel down separately or powering the entire device down by selectively removing FS_X and/or FS_R. With both channels in the standby state, power consumption is reduced to the value shown in Table 3. If transmit only operation is desired, FS_X should be applied to the device while FS_R is held low. Similarly, if receive only operation is desired, FS_R should be applied while FS_X is held low.

Fixed Data Rate Mode

Fixed data rate timing, which is 2910A and 2911A compatible, is selected by connecting $DCLK_R$ to V_{BB}. It employs master clock CLK, frame synchronization clocks FS_X and FS_R, and output $\overline{TS}_X$.

Table 3. Power-Down Methods

Device Status	Power-Down Method	Typical Power Consumption	Digital Output Status
Power Down Mode	$\overline{PDN}$ = TTL low	5 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 10 μs.
Standby Mode	FS_X and FS_R are TTL low	8 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 300 milliseconds.
Only transmit is on standby	FS_X is TTL low	50 mW	$\overline{TS}_X$ and D_X are placed in a high impedance state within 300 milliseconds.
Only receive is on standby	FS_R is TTL low	50 mW	

CLK serves as the master clock to operate the codec and filter sections and as the bit clock to clock to data in and out from the PCM highway. FS_X and FS_R are 8 KHz inputs which set the sampling frequency. $\overline{TS}_X$ is a timeslot strobe/buffer enable output which gates the PCM word onto the PCM highway when an external buffer is used to drive the line.

Data is transmitted on the highway at D_X on the first eight positive transitions of CLK following the rising edge of FS_X. Similarly, on the receive side, data is received on the first eight falling edges of CLK. The frequency of CLK must be 2.048 MHz. No other frequency of operation is allowed in the fixed data rate mode.

Variable Data Rate Mode

Variable data rate timing is selected by connecting $DCLK_R$ to the bit clock for the receive PCM highway rather than to V_{BB}. It employs master clock CLK, bit clocks $DCLK_R$ and $DCLK_X$, and frame synchronization clocks FS_R and FS_X.

Variable data rate timing allows for a flexible data frequency. It provides the ability to vary the frequency of the bit clocks, from 64 KHz to 2.048 MHz. The master clock is still restricted to 2.048 MHz.

In this mode, $DCLK_R$ and $DCLK_X$ become the data clocks for the receive and transmit PCM highways. While FS_X is high, PCM data from D_X is transmitted onto the highway on the next eight consecutive positive transitions of $DCLK_X$. Similarly, while FS_R is high, each PCM bit from the highway is received by D_R on the next eight consecutive negative transitions of $DCLK_R$.

On the transmit side, the PCM word will be repeated in all remaining timeslots in the125 μs frame as long as $DCLK_X$ is pulsed and FS_X is held high. This feature allows the PCM word to be transmitted to the PCM highway more than once per frame, if desired, and is only available in the variable data rate mode.

Precision Voltage References

No external components are required with the combochip to provide the voltage reference function. Voltage references are generated on-chip and are calibrated during the manufacturing process. These references determine the gain and dynamic range characteristics of the device.

Separate references are supplied to the transmit and receive sections and each is trimmed independently during the manufacturing process. The reference value is then further trimmed in the gain setting opamps to a final precision value. With this method the combochip can achieve the extremely accurate Digital Milliwatt Responses specified in the TRANSMISSION PARAMETERS, providing the user a significant margin for error in other board components.

TRANSMIT OPERATION

Transmit Filter

The input section provides gain adjustment in the passband by means of an on-chip operational amplifier. This operational amplifier has a common mode range of ± 2.17V, a maximum DC offset of 25 mV, and typical open loop voltage gain of 20,000. Gain of up to 20 dB can be set without degrading the performance of the filter. The load impedance to ground (GRDA) at the amplifier output (GS_X) must be greater than 10 KΩ in parallel with less than 50 pF. A DC path must be provided at VF_XI^+. The input op amp can only be used in the inverting mode as shown in Figure 3.

A low pass anti-aliasing section is included on-chip. This section typically provides 35 dB attenuation at the sampling frequency. No external components are required to provide the necessary anti-aliasing function for the switched capacitor section of the transmit filter.

The passband section provides flatness and stopband attenuation which fulfills the AT&T D3/D4 channel bank transmission specification and CCITT recommendation G.712. The 29C16 and 29C17 specifications meet or exceed digital class 5 central office switching systems requirements. The transmit filter transfer characteristics and specifications will be within the limits shown in Figure 4.

A high pass section configuration was chosen to reject low frequency noise from 50 and 60 Hz power lines, 17 Hz European electric railroads, ringing frequencies and their harmonics, and other low frequency noise. Even though there is high rejection at these frequencies, the sharpness of the band edge gives low attenuation at 200 Hz. This feature allows the use of low-cost transformer hybrids without external components.

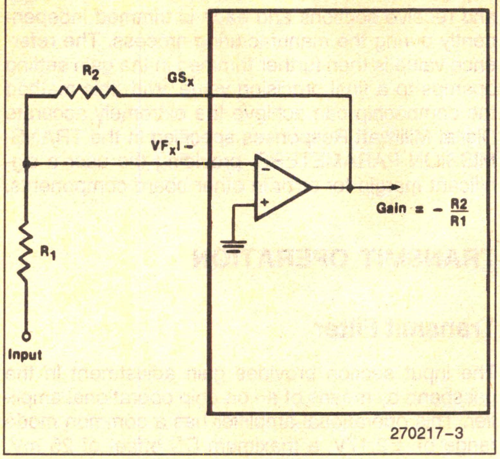


Figure 3. Transmit Filter Gain Adjustment

Encoding

The encoder internally samples the output of the transmit filter and holds each sample on an internal sample and hold capacitor. The encoder then performs an analog to digital conversion on a switched capacitor array. Digital data representing the sample is transmitted on the first eight data clock bits of the next frame.

An on-chip autozero circuit corrects for DC-offset on the input signal to the encoder. This autozero circuit uses the sign bit averaging technique; the sign bit from the encoder output is long term averaged and subtracted from the input to the encoder. In this way, all DC offset is removed from the encoder input waveform.

RECEIVE OPERATION

Decoding

The PCM word at the D_R lead is serially fetched on the first eight data clock bits of the frame. A D/A conversion is performed on the digital word and the corresponding analog sample is held on an internal sample and hold capacitor. This sample is then transferred to the receive filter.

Receive Filter

The receive filter provides passband flatness and stopband rejection which fulfills both the AT&T D3/D4 specification and CCITT recommendation G.172. The filter contains the required compensation for the (sin x)/x response of such decoders. The receive filter characteristics and specifications will be within the limits shown in Figure 5.

Receive Output Power Amplifiers

A balanced output amplifier is provided in order to allow maximum flexibility in output configuration. Either of the two outputs can be used single ended (referenced to GRDA) to drive single ended loads. Alternatively, the differential output will drive a bridged load directly. The output stage is capable of driving loads as low as 300Ω single ended or 600Ω differentially.

Transmission levels are specified relative to the receive channel output under digital milliwatt conditions, that is, when the digital input at D_R is the eight-code sequence specified in CCITT recommendation G.711.

Table 4. Zero Transmission Level Points

Symbol	Parameter	Typ	Units	Test Conditions
$0TLP1_X$	Zero Transmission Level Point Transmit Channel (0dBm0) μ-law	+2.76 +1.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_X$	Zero Transmission Level Point Transmit Channel (0dBm0) A-law	+2.79 +1.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP1_R$	Zero Transmission Level Point Receive Channel (0dBm0) μ-law	+5.76 +4.00	dBm dBm	Referenced to 600Ω Referenced to 900Ω
$0TLP2_R$	Zero Transmission Level Point Receive Channel (0dBm0) μ-law	+5.79 +4.03	dBm dBm	Referenced to 600Ω Referenced to 900Ω

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . −10°C to +80°C
Storage Temperature . . . . −65°C to +150°C
V_{CC} and GRDD
with Respect to V_{BB} . . . . −0.3V to +15V
All Input and Output Voltages
with Respect to V_{BB} . . . . −0.3V to +15V
All Input and Output Voltages
with Respect to V_{CC} . . . . −15V to +0.3V
Power Dissipation 1.35W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

NOTICE: Specifications contained within the following tables are subject to change.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, GRDA = 0V, GRDD = 0V, unless otherwise specified

Typical values are for T_A = 25°C and nominal power supply values

DIGITAL INTERFACE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{IL}	Low Level Input Current			10	μA	GRDD ≤ V_{IN} ≤ V_{IL}(1)
I_{IH}	High Level Input Current			10	μA	V_{IH} ≤ V_{IN} ≤ V_{CC}
V_{IL}	Input Low Voltage			0.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 3.2 mA at D_X, $\overline{TS_X}$
V_{OH}	Output High Voltage	2.4			V	I_{OH} = 80 μA at D_X
C_{OX}	Digital Output Capacitance(2)		5		pF	
C_{IN}	Digital Input Capacitance		5	10	pF	

POWER DISSIPATION

All measurements made at f_{DCLK} = 2.048 MHz, outputs unloaded.

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{CC1}	V_{CC} Operating Current(4)		5.6		mA	
I_{BB1}	V_{BB} Operating Current		−5.6		mA	
I_{CC0}	V_{CC} Power Down Current		0.5		mA	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
I_{BB0}	V_{BB} Power Down Current		−0.5		mA	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
I_{CCS}	V_{CC} Standby Current		0.8		mA	FS_X, FS_R ≤ V_{IL}; after 300 ms
I_{BBS}	V_{BB} Standby Current		−0.8		mA	FS_X, FS_R ≤ V_{IL}; after 300 ms
P_{D1}	Operating Power Dissipation(3)		70		mW	
P_{D0}	Power Down Dissipation(3)		5		mW	$\overline{PDN}$ ≤ V_{IL}; after 10 μs
P_{ST}	Standby Power Dissipation(3)		8		mW	FS_X, FS_R ≤ V_{IL}; after 300 ms

NOTE:
1. V_{IN} is the voltage on any digital pin.
2. Timing parameters are guaranteed based on a 100 pF load capacitance. Up to eight digital outputs may be connected to a common PCM highway without buffering, assuming a board capacitance of 60 pF.
3. With nominal power supply values.
4. V_{CC} applied last or simultaneously with V_{BB}.

ANALOG INTERFACE, TRANSMIT CHANNEL INPUT STAGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
I_{BX1}	Input Leakage Current, VF_XI-			100	nA	$-2.17V \leq V_{IN} \leq 2.17V$
R_{IXI}	Input Resistance, VF_XI-	10			MΩ	
V_{OSXI}	Input Offset Voltage, VF_XI-			25	mV	
A_{VOL}	DC Open Loop Voltage Gain, GS_X	5000				
f_c	Open Loop Unity Gain Bandwidth, GS_X		1		MHz	
C_{LXI}	Load Capacitance, GS_X			50	pF	
R_{LXI}	Minimum Load Resistance, GS_X	10			KΩ	

ANALOG INTERFACE, RECEIVE CHANNEL DRIVER AMPLIFIER STAGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
R_{ORA}	Output Resistance, PWRO+, PWRO−		1		Ω	
V_{OSRA}	Single-Ended Output DC Offset, PWRO+, PWRO−		75		mV	Relative to GRDA
C_{LRA}	Load Capacitance, PWRO+, PWRO−			100	pF	

A.C. CHARACTERISTICS—TRANSMISSION PARAMETERS

Unless otherwise noted, the analog input is a 0 dBm0, 1020 Hz sine wave.(1) Input amplifer is set for unity gain, inverting. The digital input is PCM bit stream generated by passing a 0 dBm0, 1020 Hz sine wave through an ideal encoder. Receive output is measured single ended. All output levels are (sin x)/x corrected. Typical values are for $T_A = 25°C$ and nominal power supply values.
$T_A = 0°C$ to $+70°C$; $V_{CC} = +5V \pm 5\%$; $V_{BB} = -5V \pm 5\%$; GRDA = 0; GRDD = 0; unless otherwise specified.

GAIN AND DYNAMIC RANGE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
EmW	Encoder Milliwatt Response (Transmit Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Signal input of 1.064 Vrms μ-law Signal input of 1.068 Vrms A-law $T_A = 25°C$, $V_{BB} = -5V$, $V_{CC} = +5V$
EmW_{TS}	EmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% supplies, 0 to 70°C Relative to nominal conditions
DmW	Digital Milliwatt Response (Receive Gain Tolerance)	−0.18	±0.04	+0.18	dBm0	Measure relative to $0TLP_R$. Signal input per CCITT Recommendation G.711. Output signal of 1000 Hz. $R_L = \infty$ $T_A = 25°C$; $V_{BB} = -5V$, $V_{CC} = +5V$.
DmW_{TS}	DmW Variation with Temperature and Supplies	−0.07	±0.02	+0.07	dB	±5% supplies, 0 to 70°C

NOTE:
1. 0 dBm0 is defined as the zero reference point of the channel for μ law under test (0TLP). This corresponds to an analog signal input of 1.064 Vrms or an output of 1.503 Vrms.

GAIN TRACKING Reference Level = −10 dBm0

Symbol	Parameter	2916 Min	2916 Max	2917 Min	2917 Max	Units	Test Conditions
$GT1_X$	Transmit Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2			dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT2_X$	Transmit Gain Tracking Error Sinusoidal Input; A-law				±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0
$GT1_R$	Receive Gain Tracking Error Sinusoidal Input; μ-law		±0.25 ±0.5 ±1.2			dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, R_L = 300Ω
$GT2_R$	Receive Gain Tracking Error Sinusoidal Input; A-law				±0.25 ±0.5 ±1.2	dB dB dB	+3 to −40 dBm0 −40 to −50 dBm0 −50 to −55 dBm0 Measured at PWRO+, R_L = 300Ω

NOISE All receive channel measurements are single ended

Symbol	Parameter	2916 Min	2916 Typ	2916 Max	2917 Min	2917 Typ	2917 Max	Units	Test Conditions
N_{XC1}	Transmit Noise, C-Message Weighted			15				dBrncO	Unity Gain
N_{XP}	Transmit Noise, Psophometrically Weighted						−75	dBm0p	Unity Gain
N_{RC1}	Receive Noise, C-Message Weighted: Quiet Code			11				dBrncO	D_R = 11111111
N_{RC2}	Receive Noise, C-Message Weighted: Sign bit toggle			12				dBrncO	Input to D_R is zero code with sign bit toggle at 1 KHz rate
N_{RP}	Receive Noise, Psophometrically Weighted						−79	dBm0p	D_R = lowest positive decode level
N_{SF}	Single Frequency Noise End of End Measurement			−50			−50	dBm0	CCITT G.712.4.2 Measure at PWRO+
$PSRR_1$	V_{CC} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_2$	V_{BB} Power Supply Rejection, Transmit Channel		−30			−30		dB	Idle channel; 200 mV P-P signal on supply; 0 to 50 KHz, measure at D_X
$PSRR_3$	V_{CC} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle channel; 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz

NOISE All receive channel measurements are single ended

Symbol	Parameter	2916 Min	2916 Typ	2916 Max	2917 Min	2917 Typ	2917 Max	Units	Test Conditions
$PSRR_4$	V_{BB} Power Supply Rejection, Receive Channel		−25			−25		dB	Idle channel: 200 mV P-P signal on supply; measure narrow band at PWRO+, 0 to 50 KHz
CT_{TR}	Crosstalk, Transmit to Receive			−71			−71	dB	Input = 0 dBm0, Unity Gain, 1.02 KHz, D_R = lowest positive decode level, measure at PWRO+
CT_{RT}	Crosstalk, Receive to Transmit			−71			−71	dB	D_R = 0 dBm0, 1.02 KHz, measure at D_X

DISTORTION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
$SD1_X$	Transmit Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2 (2916)	36 30 25			dB dB dB	−40 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_X$	Transmit Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2 (2917)	36 30 25			dB dB dB	−40 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD1_R$	Receive Signal to Distortion, μ-Law Sinusoidal Input; CCITT G.712-Method 2 (2916)	36 30 25			dB dB dB	−40 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
$SD2_R$	Receive Signal to Distortion, A-Law Sinusoidal Input; CCITT G.712-Method 2 (2917)	36 30 25			dB dB dB	−40 to −30 dBm0 −30 to −40 dBm0 −40 to −45 dBm0
DP_X	Transmit Single Frequency Distortion Products (29C16)			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
DP_R	Receive Single Frequency Distortion Products (29C16)			−46	dBm0	AT&T Advisory #64 (3.8) 0 dBm0 Input Signal
IMD_1	Intermodulation Distortion, End to End Measurement			−35	dB	CCITT G.712 (7.1)
IMD_2	Intermodulation Distortion, End to End Measurement			−49	dBm0	CCITT G.712 (7.2)
SOS	Spurious Out of Band Signals, End to End Measurement			−25	dBm0	CCITT G.712 (6.1)
SIS	Spurious in Band Signals, End to End Measurement			−40	dBm0	CCITT G.712 (9)
D_{AX}	Transmit Absolute Delay		245		μs	Fixed Data Rate. CLK_X = 2.048 MHz; 0 dBm0, 1.02 KHz input Signal, Unity Gain. Measure at D_X.

DISTORTION (Continued)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
D_{DX}	Transmit Differential Envelope Delay Relative to D_{AX}		170 95 45 105		μs μs μs μs	f = 500 Hz–600 Hz f = 600 Hz–1000 Hz f = 1000 Hz–2600 Hz f = 2600 Hz–2800 Hz
D_{AR}	Receive Absolute Delay		190		μs	Fixed Data Rate, CLK = 2.048 MHz; Digital Input is DMW codes. Measure at PWRO+
D_{DR}	Receive Differential Envelope Delay Relative to D_{AR}		45 35 85 110		μs μs μs μs	f = 500 Hz–600 Hz f = 600 Hz–1000 Hz f = 1000 Hz–2600 Hz f = 2600 Hz–2800 Hz

TRANSMIT CHANNEL TRANSFER CHARACTERISTICS

Input amplifier is set for unity gain, inverting.

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RX}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at VF_XI-
	16.67 Hz				dB	
	50 Hz			−30	dB	
	60 Hz			−25	dB	
	200 Hz	−1.8		−0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.10	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−32	dB	

RECEIVE CHANNEL TRANSFER CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
G_{RR}	Gain Relative to Gain at 1.02 KHz					0 dBm0 Signal Input at D_R
	Below 200 Hz			+0.125	dB	
	200 Hz	−0.5		+0.125	dB	
	300 to 3000 Hz	−0.125		+0.125	dB	
	3300 Hz	−0.35		+0.03	dB	
	3400 Hz	−0.7		−0.1	dB	
	4000 Hz			−14	dB	
	4600 Hz and Above			−30	dB	

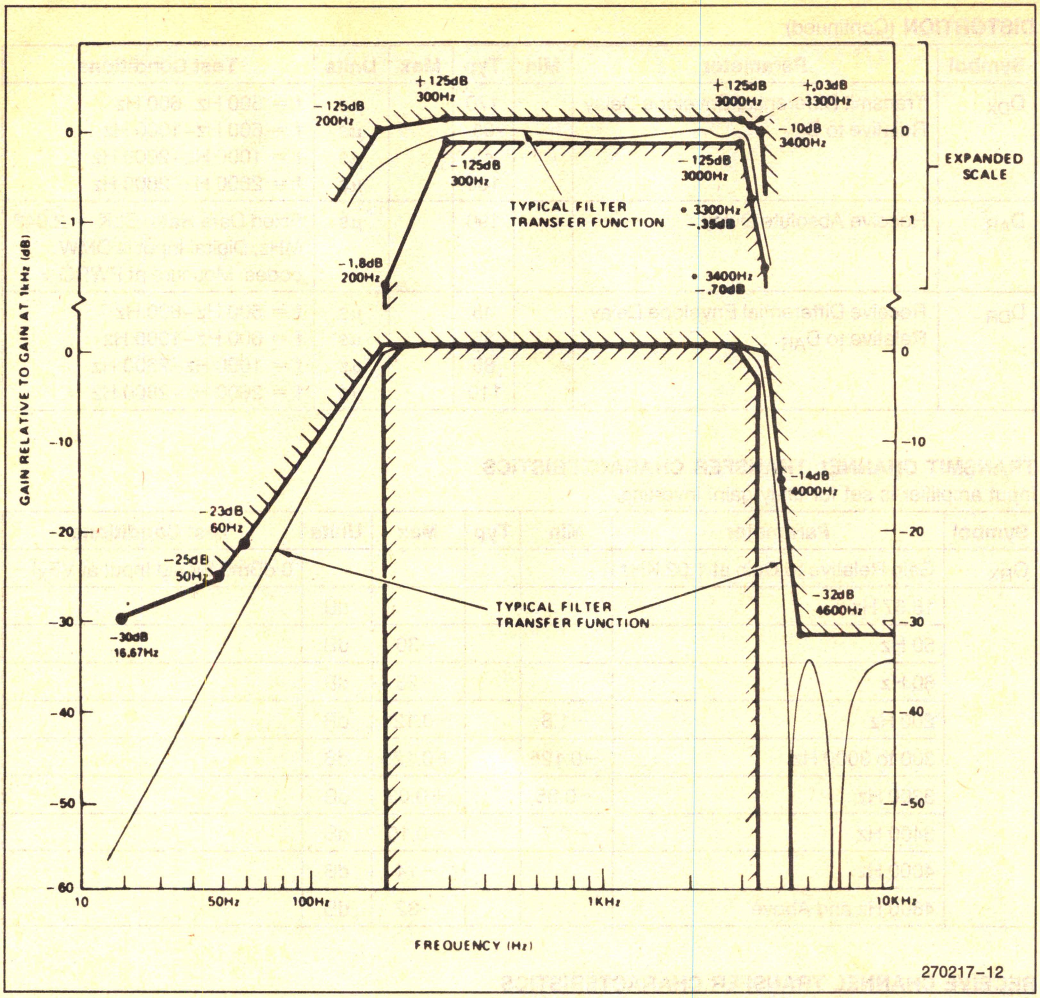


Figure 4. Transmit Channel

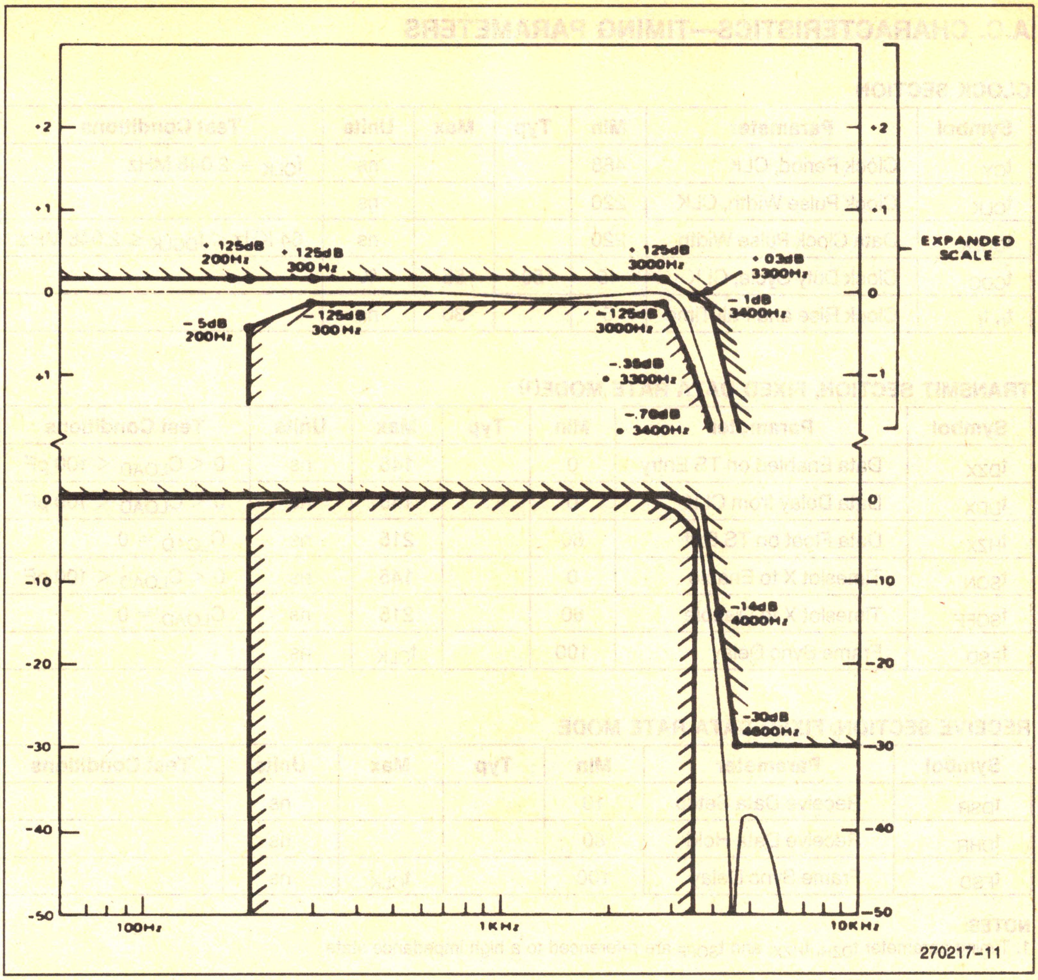


Figure 5. Receive Channel

A.C. CHARACTERISTICS—TIMING PARAMETERS

CLOCK SECTION

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{CY}	Clock Period, CLK	488			ns	$f_{CLK} = 2.048$ MHz
t_{CLK}	Clock Pulse Width, CLK	220			ns	
t_{DCLK}	Data Clock Pulse Width	220			ns	64 KHz $\leq f_{DCLK} \leq$ 2.048 MHz
t_{CDC}	Clock Duty Cycle, CLK	45	50	55	%	
t_r, t_f	Clock Rise and Fall Time	5		30	ns	

TRANSMIT SECTION, FIXED DATA RATE MODE(1)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{DZX}	Data Enabled on TS Entry	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{DDX}	Data Delay from CLK	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{HZX}	Data Float on TS Exit	60		215	ns	$C_{LOAD} = 0$
t_{SON}	Timeslot X to Enable	0		145	ns	$0 < C_{LOAD} < 100$ pF
t_{SOFF}	Timeslot X to Disable	60		215	ns	$C_{LOAD} = 0$
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	

RECEIVE SECTION, FIXED DATA RATE MODE

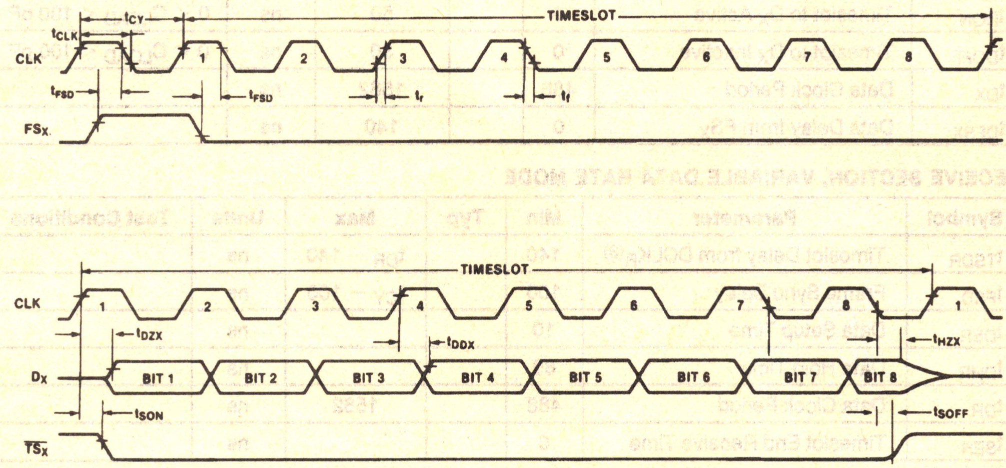
Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{DSR}	Receive Data Setup	10			ns	
t_{DHR}	Receive Data Hold	60			ns	
t_{FSD}	Frame Sync Delay	100		t_{CLK}	ns	

NOTES:
1. Timing parameter t_{DZH}, t_{HZX}, and t_{SOFF} are referenced to a high impedance state.

WAVEFORMS

Fixed Data Rate Timing

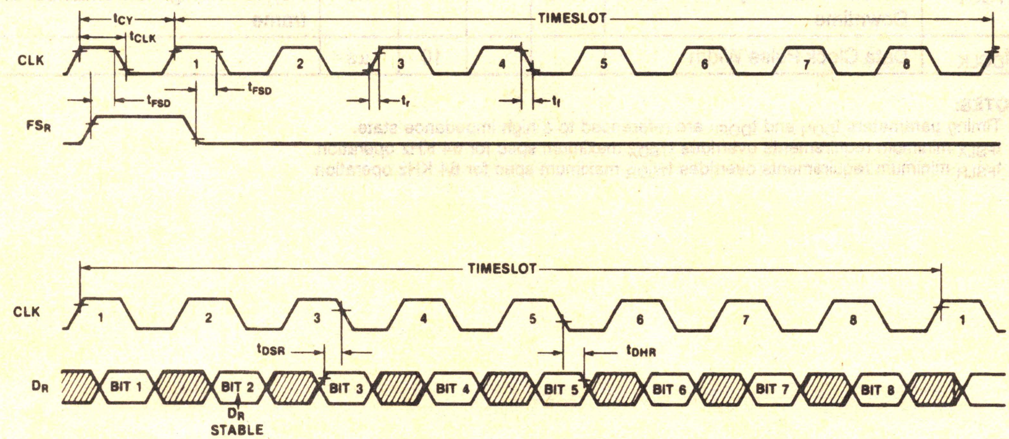
TRANSMIT TIMING



270217-6

NOTE:
All timing parameters referenced to V_{IH} and V_{IL} except t_{DZX}, t_{SOFF} and t_{HZX} which reference a high impedance state

RECEIVE TIMING



270217-7

NOTE:
All timing parameters referenced to V_{IH} and V_{IL}.

TRANSMIT SECTION, VARIABLE DATA RATE MODE(1)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{TSDX}	Timeslot Delay from $DCLK_X$(2)	140		$t_{DX} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DDX}	Data Delay from $DCLK_X$	0		100	ns	$0 < C_{LOAD} < 100$ pF
t_{DON}	Timeslot to D_X Active	0		50	ns	$0 < C_{LOAD} < 100$ pF
t_{DOFF}	Timeslot to D_X Inactive	0		80	ns	$0 < C_{LOAD} < 100$ pF
t_{DX}	Data Clock Period	488		1562	ns	
t_{DFSX}	Data Delay from FS_X	0		140	ns	

RECEIVE SECTION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{TSDR}	Timeslot Delay from $DCLK_R$(3)	140		$t_{DR} - 140$	ns	
t_{FSD}	Frame Sync Delay	100		$t_{CY} - 100$	ns	
t_{DSR}	Data Setup Time	10			ns	
t_{DHR}	Data Hold Time	60			ns	
t_{DR}	Data Clock Period	488		1562	ns	
t_{SER}	Timeslot End Receive Time	0			ns	

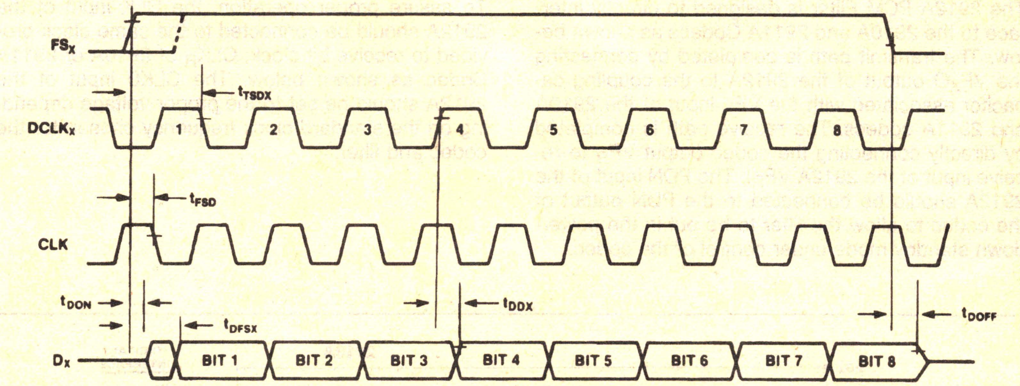
64 KB OPERATION, VARIABLE DATA RATE MODE

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
t_{FSLX}	Transmit Frame Sync Minimum Downtime	488			ns	FS_X is TTL high for remainder of frame
t_{FSLR}	Receive Frame Sync Minimum Downtime	1952			ns	FS_R is TTL high for remainder of frame
t_{DCLK}	Data Clock Pulse Width			10	μs	

NOTES:
1. Timing parameters t_{DON} and t_{DOFF} are referenced to a high impedance state.
2. t_{FSLX} minimum requirements overrides t_{TSDX} maximum spec for 64 KHz operation.
3. t_{FSLR} minimum requirements overrides t_{TSDR} maximum spec for 64 KHz operation.

VARIABLE DATA RATE TIMING

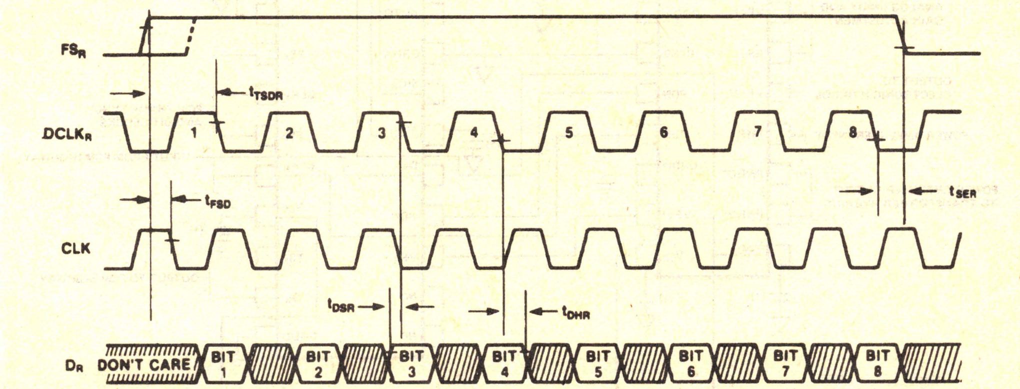
TRANSMIT TIMING



NOTE: 270217-8

All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

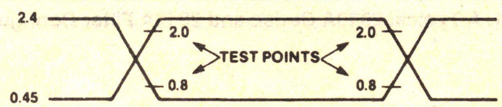
RECEIVE TIMING



NOTE: 270217-9

All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM



270217-10

A.C. testing inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 2.0V for a Logic "1" and 0.8V for a Logic "0".

intel®

APPLICATIONS INFORMATION
2910A/2911A/2912A

CODEC INTERFACE

The 2912A PCM Filter is designed to directly interface to the 2910A and 2911A Codecs as shown below. The transmit path is completed by connecting the VF_XO output of the 2912A to the coupling capacitor associated with the VF_X input of the 2910A and 2911A codecs. The receive path is completed by directly connecting the codec output VF_R to receive input of the 2912A VF_RI. The PDN input of the 2912A should be connected to the PDN output of the codec to allow the filter to be put in the power-down standby mode under control of the codec.

CLOCK INTERFACE

To assure proper operation, the CLK input of the 2912A should be connected to the same clock provided to receive bit clock, CLK_R of 2910A or 2911A Codec as shown below. The CLK0 input of the 2912A should be set to the proper voltage depending on the standard clock frequency chosen for the codec and filter.

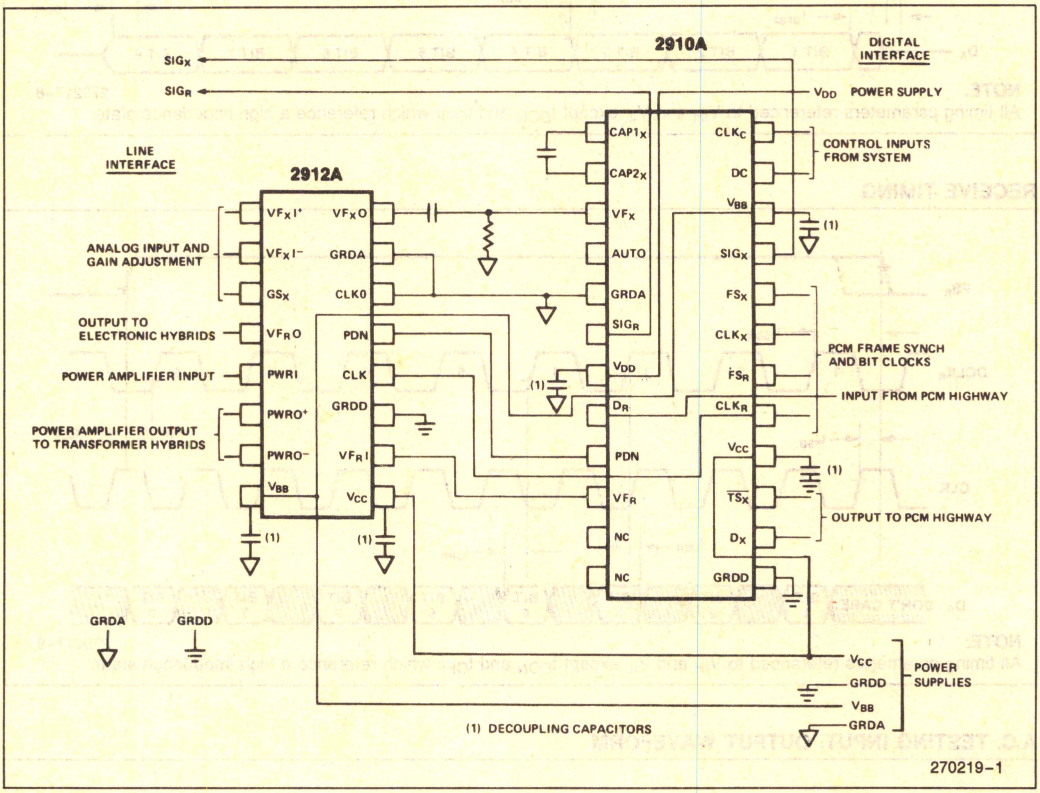


Figure 1. A Typical 2910A Codec and 2912A Filter Configuration

October 1986
Order Number: 270219-001

GROUNDING, DECOUPLING, AND LAYOUT RECOMMENDATIONS

The most important steps in designing a low noise line card are to insure that the layout of the circuit components and traces results in a minimum of cross coupling between analog and digital signals, and to provide well bypassed and clean power supplies, solid ground planes, and minimal lead lengths between components.

1) All power source leads should be bypassed to ground on each printed circuit board (PCB), on which codecs are provided. At least one electrolytic bypass capacitor (at least 50 μF) per board is recommended at the point where all power traces from the codecs and filters join prior to interfacing with the edge connector pins assigned to the power leads.
2) When using two-sided PCBs, use both corresponding pins on opposite sides of the board for the same power lead. Strap them together both on the PCB and on the back of the edge connector.
3) Lay out the traces on codec- and filter-equipped boards such that analog signal and capacitor leads are separated as widely as possible from the digital clock and data leads.
4) Connect the codec sample and hold capacitor with the shortest leads possible. Mount it as close to the codec CAP1X, CAP2X pins as possible. Shield the capacitor traces with analog ground.
5) Do not lay out any board traces (especially digital) that pass between or near the leads of the sample and hold capacitor(s) since they are in high impedance circuits which are sensitive to noise coupling.
6) Keep analog voice circuit leads paired on their layouts so that no intervening circuit leads are permitted to run parallel to them and/or between them.
7) Arrange the layout for each duplicated line, trunk or channel circuit in identical form.
8) Line circuits mounted extremely close to adjacent line circuits increase the possibility of interchannel crosstalk.
9) Avoid assignment of edge connector pins to any analog signal adjacent to any lead carrying digital (periodic) signals or power.
10) The optimum grounding configuration is to maintain separate digital and analog grounds on the circuit boards, and to carry these grounds back to the power supply with a low impedance connection. This keeps the grounds separate over the entire system except at the power supply.
11) The voltage difference between ground leads GRDA and GRDD (analog and digital ground) should not exceed two volts. One method of preventing any substantial voltage difference between leads GRDA and GRDD is to connect two diodes back to back in opposite directions across these two ground leads on each board.
12) Codec-filter pairs should be aligned so that pins 9 through 16 of the filter face pins 1 through 12 of the codec. This minimizes the distance for analog connections between devices and with no crossing analog lines.
13) No digital or high voltage level (such as ringing supply) lines should run under or in parallel with these analog VF connections. If the analog lines are on the top (component side) of the PC board, then GRDD, GRDA, or power supply leads should be directly under them, on the bottom to prevent analog/digital coupling.
14) Both the codec and filter devices should be shielded from traces on the bottom of the PCB by using ground or power supply leads on the top side directly under the device (like a ground plane).
15) Two +5V power supply leads (V_{CC}) should be used on each PCB, one to the filters, the other to the codecs. These leads should be separately decoupled at the PCB where they then join to a single 5V supply at the backplane connector. Decoupling can be accomplished with either a series resistor/parallel capacitor (RC lowpass) or a series RF choke and parallel capacitor of each 5V lead. The capacitor should be at least 10 μF in parallel with a 0.1 μF ceramic. This filters both high and low frequencies and accommodates large current spikes due to switching.
16) Both grounds and power supply leads must have low resistance and inductance. This should be accomplished by using a ground plane whenever possible. When narrower traces must be used, a minimum width of 4 millimeters should be maintained. Either multiple or extra large plated through holes should be used when passing the ground connections through the PCB.

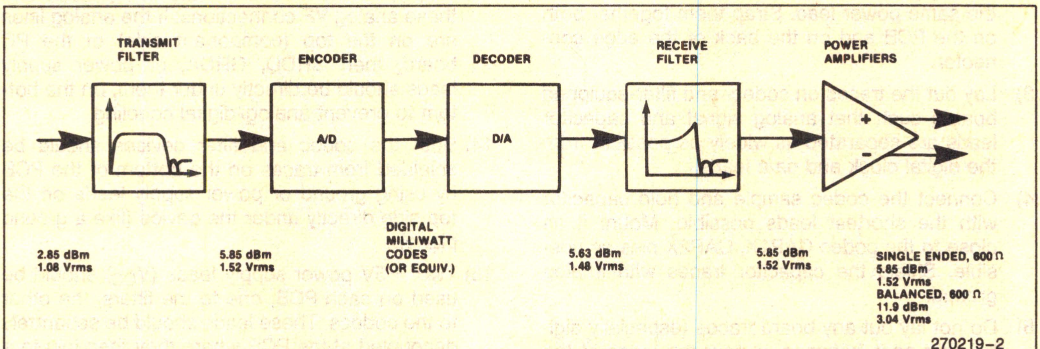
17) The 2912A PCM filter should have all power supplies bypassed to analog ground (GRDA). The 2910A/2911A Codec +5V power supplies should be bypassed to the digital ground (GRDD). This is appropriate when separate +5V power supply leads are used as suggested in item 15. The −5V and +12V supplies should be bypassed to analog ground (GRDA). Bypass capacitors at each device should be high frequency capacitors of approximately 0.1 to 1.0 μF value. Their lead lengths should be minimized by routing the capacitor leads to the appropriate ground plane under the device (either GRDA or GRDD).

18) Relay operation, ring voltage application, interruptions, and loop current surges can produce enormous transients. Leads carrying such signals must be routed well away from both analog and digital circuits on the line card and in backplanes. Lead pairs carrying current surges should be routed closely together to minimize possible inductive coupling. The microcomputer clock lead is particularly vulnerable, and should be buffered. Care should also be used in the backplane layout to prevent pickup surges. Any other latching components (relay buffers, etc.) should also be protected from surges.

19) When not used, the AUTO pin should float with minimum PC board track area.

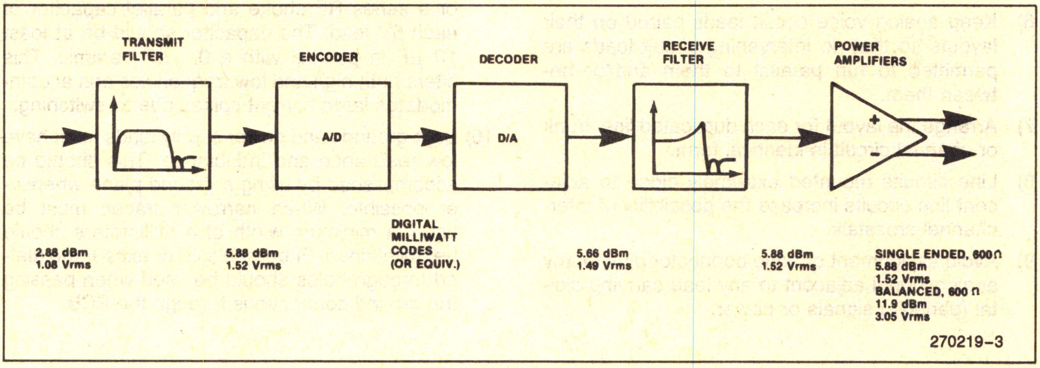
ZERO TRANSMISSION LEVEL POINTS

2910A/2912A 0 dBm0



270219-2

2911A/2912A 0 dBm0



270219-3

intel®

APPLICATION
NOTE

AP-142

November 1986

Designing Second-Generation Digital Telephony Systems Using the Intel 2913/14 Codec/Filter Combochip

ROBERT E. HOLM
TELECOM TECHNICAL SUPPORT

JOHN HUGGINS
TELECOM DESIGN ENGINEERING

Order Number: 210314-002

Note: See data sheet for latest specifications. Values given in this application note are for reference only, and were considered correct at the time of publication (Feb. 1982).

1.0 INTRODUCTION

This application note describes the features and capabilities of the 2913 and 2914 codec/filter combochips, and relates these capabilities to the design and manufacturing of transmission and switching linecards.

1.1 Background

The first generation of per line codecs (Intel 2910A/11A) and filters (Intel 2912A) economically integrated the analog-digital conversion circuits and PCM formatting circuits into one chip and the filtering and gain setting circuits into another chip. These two chips helped to make possible the rapid conversion to digital switching systems that has taken place in the last few years.

The second generation of Intel LSI PCM telephony components, the 2913/14 Combochip, extends the level of integration of the linecard by combining the codec and filter functions for each line on a single LSI chip. In the process of combining both functions, circuit design improvements have also improved performance, reduced external component count, lowered power dissipation, increased reliability, added new features, and maintained architectural transparency.

The 2913 and 2914 data sheet contains a complete description of both parts, including detailed discussions of each feature and specifications for timing and performance levels. This application note, in conjunction with the data sheet, describes in more detail how the new and improved features help in the design of second-generation linecards first by comparing the two generations of components to see where the improvements have been made, and then by discussing specific design considerations.

1.2 Comparison of First- and Second-Generation Component Capabilities

The combochip represents a higher level of component integration than the devices it replaces and, because of the economics of LSI (replacing two chips with one), ultimately will cost significantly less at the component level. But comparison of the combochip block diagram with first-generation single-chip codec and filter reveals few major functional differences. Figure 1 compares the first-generation codec and filter chips to the combochip. Both provide rigidly specified PCM capabilities of voice signal bandlimiting and nonlinear companded A/D and D/A conversion. The first on-chip reference voltage was introduced in the 2910/2911 single-chip codecs and is included in the combochip. The provision of uncommitted buffer amplifiers for flexible transmission level adjustment and enhanced analog output drive was a feature of the now standard 2912 switched-capacitor PCM filter is available on the combochip. Like-

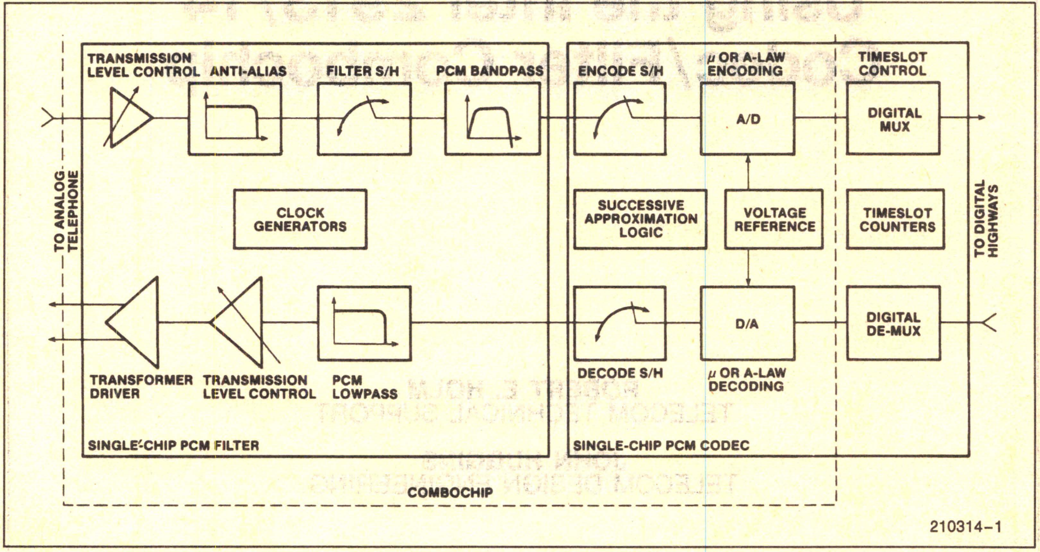


Figure 1. LSI Partitioning of Codec/Filter Functions

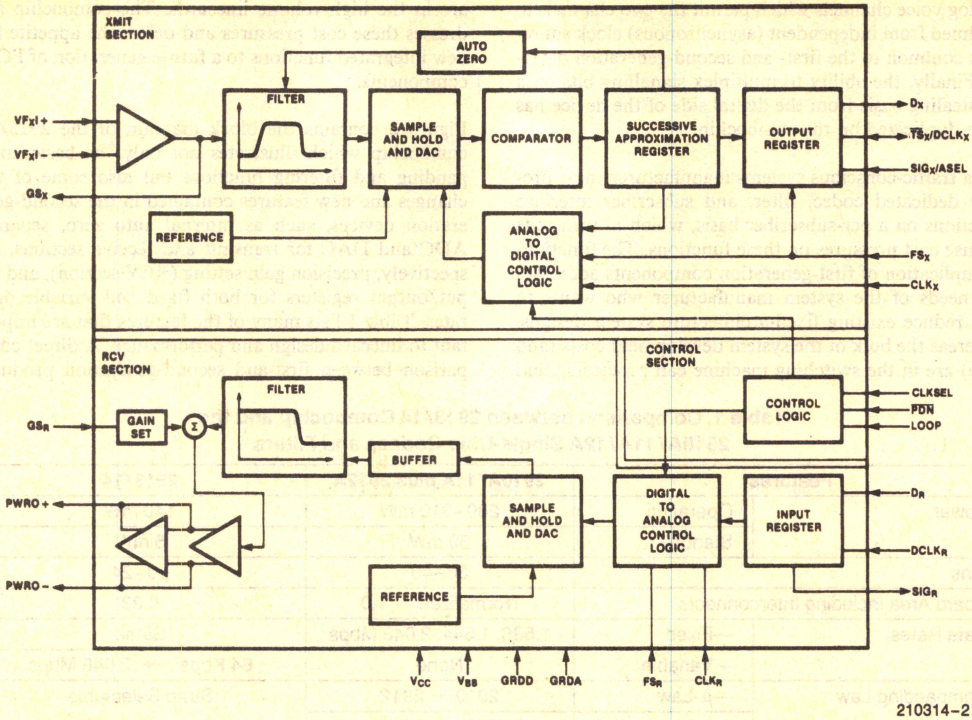
wise, independent transmit (A/D) and receive (D/A) analog voice channels which permit the two channels to be timed from independent (asynchronous) clock sources is common to the first- and second-generation devices. Finally, the ability to multiplex signalling bits on a bit-stealing basis from the digital side of the device has been duplicated on the combochip.

Data traffic-conscious systems manufacturers now provide dedicated codec, filter, and subscriber interface functions on a per-subscriber basis, which in turn puts intense cost pressures on these functions. The functional duplication of first-generation components addresses the needs of the system manufacturer who wants to cost reduce existing fixed-architecture system designs. Whereas the bulk of the system development costs (and time) are in the switching machine call processing and diagnostic software, the bulk of the production costs are in the high-volume linecards. The combochip addresses these cost pressures and defers the appetite for new integrated functions to a future generation of PCM components.

Figure 2 contains the block diagram of the 2913/14 combochip which illustrates not only the basic companding and filtering functions but also some of the changes and new features contained in the second-generation devices, such as internal auto zero, separate ADC and DAC for transmit and receive sections, respectively, precision gain setting (RCV section), and input/output registers for both fixed and variable data rates. Table 1 lists many of the features that are important to linecard design and performance. A direct comparison between first-and second-generation products

Table 1. Comparison between 2913/14 Combochip and the 2910A/11A/12A Single-Chip Codecs and Filters

Features		2910A/11A plus 2912A	2913/14
Power	Operating	280–310 mW	140 mW
	Standby	33 mW	5 mW
Pins		38–40	20–24
Board Area Including Interconnects		Normalized = 1.0	0.33
Data Rates	—Fixed	1.536, 1.544, 2.048 Mbps	Same
	—Variable	None	64 Kbps → 2.048 Mbps
Companding Law	—μ-Law	2910 + 2912	Strap Selectable
	—A-Law	2911 + 2912	
PSRR	1 KHz	30 dB	>35 dB
	> 10 KHz	Not Spec'd	> 35 dB
Gain Setting		Trim Using Pot Necessary	Precision Resistors Eliminate Trim Req.
Operating Modes	Direct	Yes	Yes
	Timeslot Assign	Yes	No
On-Chip V_{REF}		Yes	Yes
ICN — Half Channel Improvement		15 dBrnC0 Transmit 11 dBrnC0 Receive	15 dBrnC0 Transmit 11 dBrnC0 Receive
S/D — Half Channel Improvement		See Data Sheet	See Section 2.0
GT — Half Channel Improvement		See Data Sheet	See Section 2.0
Power Down (Standby)		PDN Pin	Frame Sync Removal or PDN Pin
Signalling		2910-8th Bit	2914-8th Bit
Auto Zero		External	Internal
S & H Caps		External Transmit Internal Receive	Internal
Test Modes		None	Design Tests Manufacturing Test On-Line Operational Tests
Encoder Implementation		Resistive Ladder	Capacitive Charge Redistribution Ladder
Filter/Gain Trim		Fuse Blowing ± 0.2 dB	Fuse Blowing ± 0.04 dB



(a) Combochip Block Diagram

V_{BB}	Power (−5V)	GS_X	Transmit Gain Control
PWRO+, PWRO−	Power Amplifier Outputs	VF_XI-, VF_XI+	Analog Inputs
GS_R	Receive Gain Control	GRDA	Analog Ground
$\overline{PDN}$	Power Down Select	NC	No Connect
CLKSEL	Master Clock Frequency Select	SIG_X	Transmit Signaling Input
LOOP	Analog Loop Back	ASEL	μ- or A-law Select
SIG_R	Receive Signaling Bit Output	$\overline{TS_X}$	Timeslot Strobe/Buffer Enable
$DCLK_R$	Receive Variable Data Clock	$DCLK_X$	Transmit Variable Data Clock
D_R	Receive PCM Input	D_X	Transmit PCM Output
FS_R	Receive Frame Synchronization Clock	FS_X	Transmit Frame Synchronization Clock
GRDD	Digital Ground	CLK_X	Transmit Master Clock
V_{CC}	Power (+5V)	CLK_R	Receive Master Clock

(b) Combochip Pin Names

Figure 2. Block Diagram of 2913/14 Combochip

shows the significant improvement in the combochip both in performance levels and system flexibility.

2.0 DESIGN CONSIDERATIONS

The key point with the 2913/14 is that it will result in a linecard that performs better and costs less than any two-chip codec/filter solution. The lower cost results from many factors, as seen in Table 2. Both direct replacement costs and less tangible design and manufacturing time savings combine to yield lower recurring and nonrecurring costs. As an example, the wider margins to transmission specs and the higher power supply rejection ratios of the 2913/14 will both shorten the design time needed to build and test the linecard prototype and reduce the reject rate on the manufacturing line.

Table 2. 2913/14 Factors which Lower the Cost of Linecard Design and Manufacturing

- Lower LSI Cost (2914 vs. 2910/11 + 2912)
- Fewer External Components
- Less Board Area
- Shorter Design/Prototype Cycle
- Better Yields/Higher Reliability
- Lower Power/Higher Density

Part of the recurring cost of linecard production is the efficiency of the manufacturing line in turning out each board. This is measured in both parts cost and time. Average manufacturing time is strongly effected by the line yield, i.e., the reject rate reliability. A linecard using the 2913/14 has many labor-saving features, which also increases the *reliability* of the manufacturing process. Some of these features are detailed in Table 3.

The combination of fewer parameters to trim (gain, reference voltage, etc.), tolerance to wider power supply variations, and on-chip test modes make the linecard very manufacturable compared to first-generation designs.

Probably the most obvious improvement in linecard design based around the 2913/14 is the reduction in linecard PCB area needed compared to two-chip designs. The combination of the codec and filter into a single package alone reduced the LSI area by one-third. Table 4 shows many of the other ways in which board area is conserved. In general, it reduces to fewer components, more on-chip features, and layout of the chip resulting in an efficient board layout which neatly separates the analog and digital signals both inside the chip and on the board.

Table 3. 2914 Factors which Increase Linecard Manufacturing Yields and Efficiency

- Higher Reliability
  - —Fewer connections and components
  - —More integrated packaging
  - —More margin to specs
  - —Lower power
  - —NMOS proven process
  - —Less sensitive to parameter variations
- Fewer Manufacturing Steps
  - —No gain trimming
  - —On chip V_{REF}
  - —Wide power supply tolerance
  - —On chip test modes
  - —Wide margins to spec

Table 4. Design Factors for 2914 which Reduce Linecard PCB Area

- Integrated Packaging
  - —2914 vs. 2910/11 + 2912 = 1/3 board area
  - —2913 takes even less space
- Fewer Interconnects/Components
  - —Codec/filter combined
  - —On-chip reference voltage
  - —On-chip auto zero
  - —On-chip capacitors
  - —No gain trim components
  - —No voltage regulators
- Efficient Layout (Facilitates Auto Insertion)
  - —Analog/digital sections separated on chip
  - —Digital traces can cross under chip
  - —Two power supplies only
  - —Low power/high density

Table 5. 2913/14 Operating Mode Options Add Flexibility to Linecard Design

Option	Mode Control Pins	Results of Mode Selection	
		2914 (24 Pin)	2913 (20 Pin)
Companding Law	SIGX/ASEL	A-Law or μ-Law + Signalling	A-Law/μ-Law, no Signalling
Power Down	$\overline{PDN}$	Transmit & Receive Side Go To Standby Power (5 mW)	
	FS_X & FS_R Removed	Same (12 mW)	
	FS_X Removed	Transmit Side Goes to Standby (110 mW)	
	FS_R Removed	Receive Side Goes to Standby (70 mW)	
Data Rate	= V_{CC}/GRDD/V_{BB} $DCLK_R = V_{BB}$	1.536/1.544/2.048 Mbps in Fixed Data Rate Mode	
	= V_{CC}/GRDD/V_{BB} $DCLK_B$ = Clock	Variable Data Rate Mode from 64 Kbps to 2.048 Mbps, No Signalling	
Test Modes	LOOP = V_{CC}	Implements Analog Loopback	No Loopback Capability
	$\overline{PDN} = V_{BB}$	Provides Access to Transmit Codec Through ASEL and $\overline{TSX}$ Pins	
	$D_R = V_{BB}$	Provides Access to RCV Filter Input at $DCLK_R$ and Transmit Filter Outputs at ASEL and $\overline{TSX}$ Pins	

Many of the factors discussed—which result in efficient, cost-effective linecard designs—are discussed in more detail both in the 2913/14 data sheet and in the following sections of this note.

2.1 Operating and Test Mode Selection

A key to designing with the 2913/14 combo is the wide range of options available in configuring, either with strap options or in real time, the different modes of operation. The 2913 combochip (20 pins) is specifically aimed at synchronous switching systems (remote concentrators, PABXs, central offices) where small package size is especially desirable. The 2914 combochip (24 pins) has additional features which are most suitable for applications requiring 8th-bit signalling, asynchronous operation, and remote testing of transmission paths (e.g., channel banks). Once the specific device is selected, there is a wide range of operating modes to use in the card design, as seen in Table 5. This table lists the optional parameters and the pins which control the operating mode. The result of selecting a mode is listed for both the 2913 and 2914.

The purpose of offering these options is to ensure that the 2913/14 combo will accommodate any existing linecard design with architectural transparency. At the same time, features were designed in to facilitate design and manufacturing testing to reduce overall cost of development and production.

2.2 Data Rate Modes

Any rapid conversion scenario presumes that the combochip will fit existing system architectures (retrofit) without significant system timing, control, or software modifications. To this end, two distinct user-selectable timing modes are possible with the combochip. For purposes of discussion, these are designated (a) fixed data rate timing (FDRT) and (b) variable data rate timing (VDRT).

FDRT is identical to the 2910/2911 codec timing in which a single high-speed clock serves both as master clock for the codec/filter internal conversion/filtering functions and as PCM bit clock for the high-speed serial PCM data bus over which the combochip transmits and receives its digitized voice code words. In this mode, PCM bit rates are necessarily confined to one of three distinct frequencies (1.536 MHz, 1.544 MHz, or 2.048 MHz). Many recently designed systems employ this type of timing which is sometimes referred to as burst-mode timing because of the low duty cycle of each timeslot (i.e., channel) on the time division multiplexed PCM bus. It is possible for up to 32 active combochips to share the same serial PCM bus with FDRT.

VDRT (sometimes referred to as shift register timing), by comparison, utilizes one high-speed master clock for the combochip internal conversion/filtering functions and a separate, variable frequency, clock as the PCM bit clock for the serial PCM data bus. Because the serial PCM data rate is independent of internal conversion timing, there is considerable flexibility in the choice of PCM data rate. In this mode the master clock is permitted to be 1.536 MHz, 1.544 MHz, or 2.048 MHz, while the bit clock can be any rate between 64 KHz and 2.048 MHz. In this mode it is possible to have a dedicated serial bus for each combochip or to share a single serial PCM bus among as many as 32 active combochips.

Thus, the two predominant timing configurations of present system architectures are served by the same device, allowing, in many cases, linecard redesign without modification of any common system hardware or software. Additional details relating to the design of systems using either mode are found in section 3.0.

2.3 Margin to Performance Specifications

The combochip benefits from design, manufacturing, and test experience with first-generation PCM products on the part of the system manufacturer, component suppliers, and test equipment suppliers. The sub-millivolt PCM measurement levels and tens of microvolts accuracy requirements on the lowest signal measurements often result in tester correlation problems, yield losses, and excess costs for system and PCM component manufacturers alike. Thus additional performance margin built into the PCM components themselves will have its effect on line circuit costs even though the system transmission specifications may not reflect the improved performance margin.

Half channel measurements have been made of the transmission parameters—gain tracking (GT), signal to distortion ratio (S/D), and idle channel noise (ICN).

Gain Tracking—Figure 3 shows the gain tracking data for both the transmit and receive sides of the combo using both sine wave testing (CCITT G712.11 Method 2) and white noise testing (CCITT G712.11 Method 1). The data shows a performance very nearly equal to the theoretically best achievable using both test techniques. End to end measurements, although not spec'd, also show a corresponding good performance with errors less than or equal to the sum of the half channel values.

Signal to Distortion Ratio—This is a measure of the system linearity and the accuracy in implementing the companding codes. Figure 4 shows the excellent perfor-

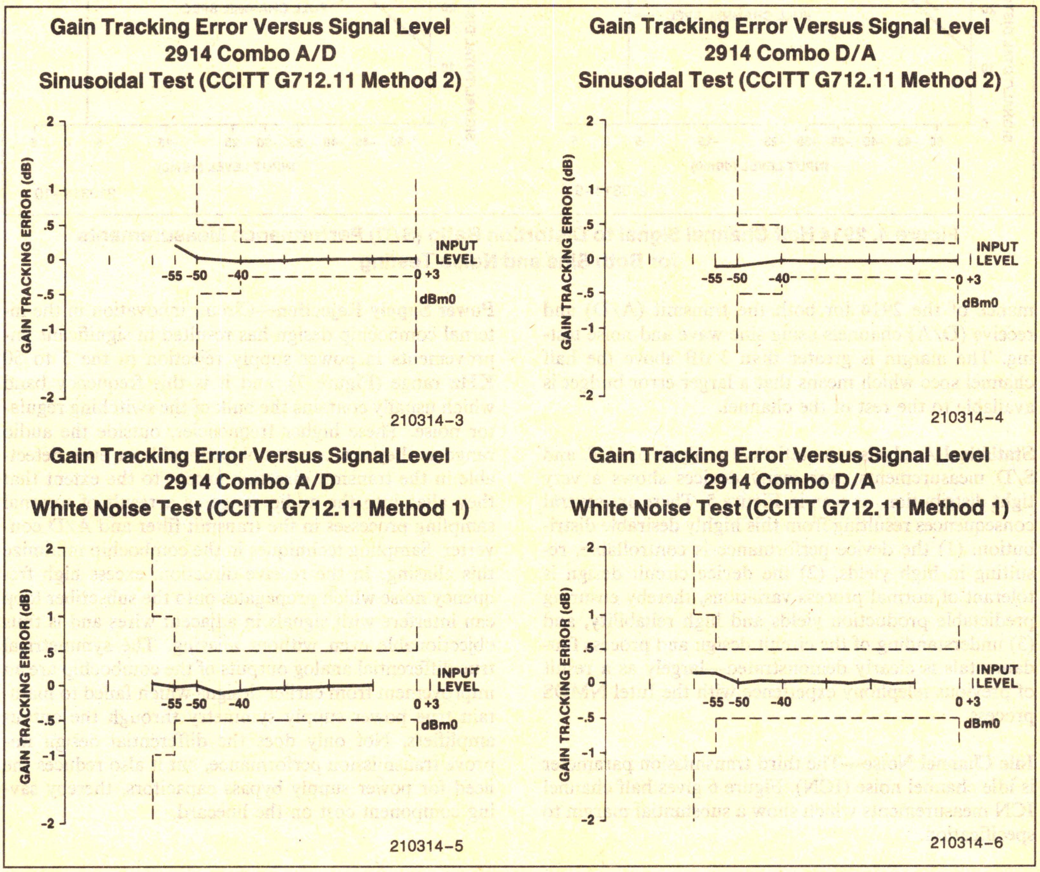


Figure 3. 2914 Half Channel Gain Tracking Performance Measurements for Both Sine and Noise Testing

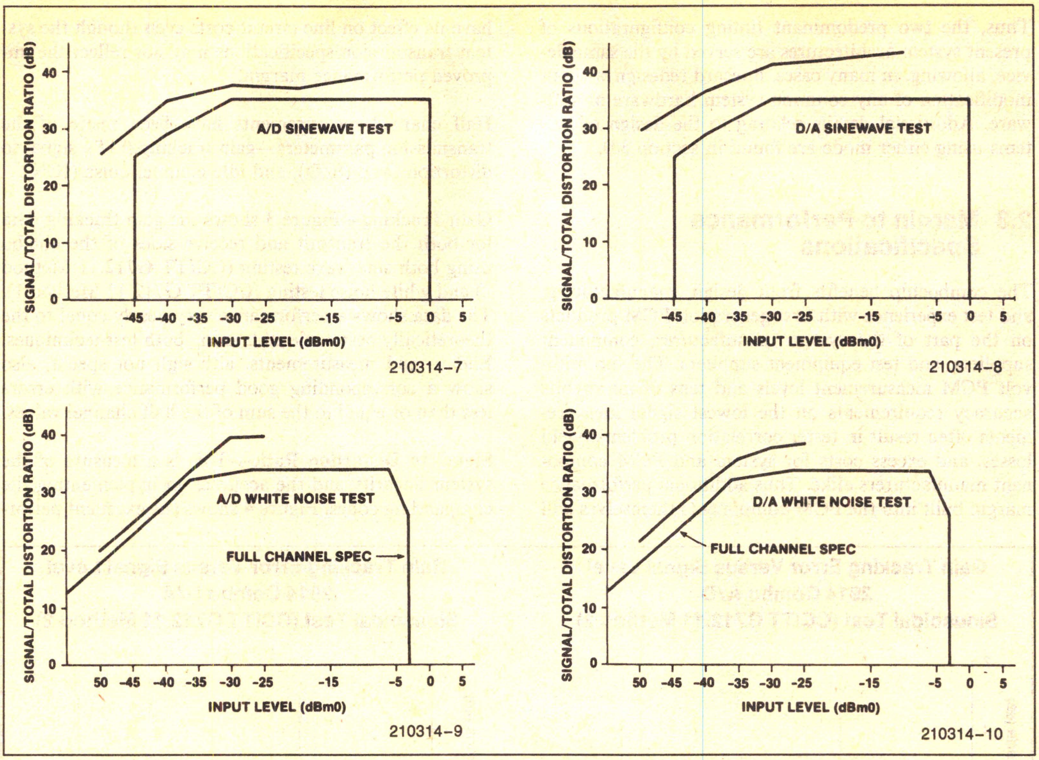


Figure 4. 2914 Half Channel Signal to Distortion Ratio (S/D) Performance Measurements for Both Sine and Noise Testing

mance of the 2914 for both the transmit (A/D) and receive (D/A) channels using sine wave and noise testing. The margin is greater than 3 dB above the half channel spec which means that a larger error budget is available to the rest of the channel.

Statistical Analysis—A statistical analysis of G.T. and S/D measurements over many devices shows a very tight distribution, as seen in Figure 5. There are several consequences resulting from this highly desirable distribution: (1) the device performance is controllable, resulting in high yields, (2) the device circuit design is tolerant of normal process variations, thereby ensuring predictable production yields and high reliability, and (3) understanding of the circuit design and process fundamentals is clearly demonstrated—largely as a result of previous telephony experience with the Intel NMOS process.

Idle Channel Noise—The third transmission parameter is idle channel noise (ICN). Figure 6 gives half channel ICN measurements which show a substantial margin to specification.

Power Supply Rejection—Circuit innovation in the internal combochip design has resulted in significant improvements in power supply rejection in the 5 to 50 KHz range (Figure 7), and it is this frequency band which usually contains the bulk of the switching regulator noise. These higher frequencies, outside the audio range as they are, are not objectionable or even detectable in the transmit direction except to the extent that they alias into the audio range as a result of internal sampling processes in the transmit filter and A/D converter. Sampling techniques in the combochip minimize this aliasing. In the receive direction, excess high frequency noise which propagates onto the subscriber loop can interfere with signals in adjacent wires and is thus objectionable even without aliasing. The symmetrical true differential analog outputs of the combochip are an improvement from earlier designs which failed to maintain true power supply symmetry through the output amplifiers. Not only does the differential design improve transmission performance, but it also reduces the need for power supply bypass capacitors, thereby saving component cost on the linecard.

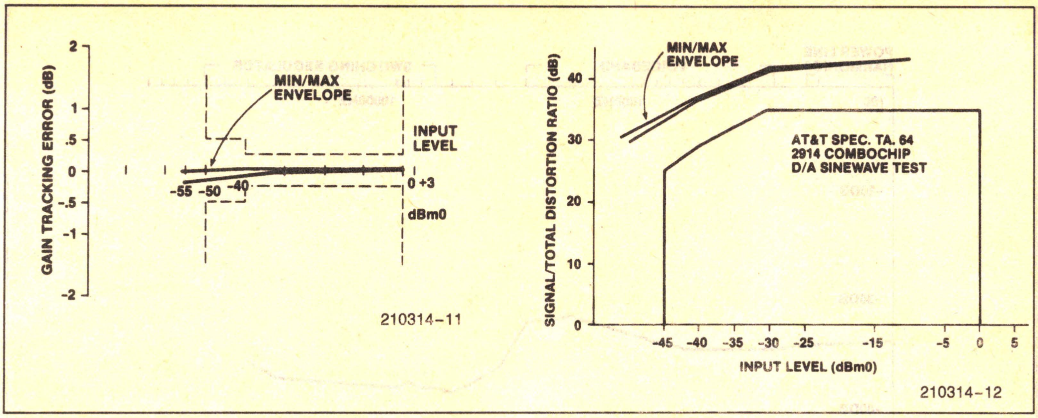


Figure 5. Statistical Analysis of Transmission Performance Showing Tight Distribution Over Many Devices

	Weighting	ICN
A/D	C Message	15 dBrnC0
D/A	C Message	11 dBrnC0

Figure 6. 2914 Idle Channel Noise (ICN) Measurements

Autozero—The autozero circuit is contained completely on-chip. It automatically centers the signal/noise distribution at the encoder input. This ensures minimal ICN due to bit toggling and also maintains maximum sensitivity to the AC signals of interest.

2.4 Power Conservation

Figure 8 illustrates typical power consumption and office equipment dissipation for a resistive line biasing arrangement (with no loop current limiting) and for the per-line PCM components. It can be seen that overall line circuit power consumption and dissipation are strong functions of subscriber loop resistance, and are dominated by line biasing current regardless of loop length. It can also be seen that the combochip achieves significant reductions in PCM component contributions relative to both the 2910A/2912A and 2910/2912. Present residential traffic characteristics are such that the PCM components are active less than 10% of the time, and in its low-power standby state, the combochip power dissipation drops to typically 5 mW as the line current (and dissipation) goes to its background on-hook leakage level of typically a few milliwatts (but for very leaky lines, as much as 50 mW–500 mW).

The concern for linecard power consumption and dissipation is related both to the cost of providing power and to the system density problem involving convection heat removal from the linecards. Consequently, much recent line circuit development activity centers on elimination of the inefficient resistive line current feed both by current limiting in short loops and by more exotic and expensive per-line dc-dc converters. For both present-generation designs and cost-reduction redesigns, the typical combochip dissipation of 140 mW active/5 mW standby will allow system board packing density improvements and power supply cost reductions.

A closer look at the effect of loading (duty cycle) on the average power dissipation of a combochip is given in Table 6. Typical loading percents run as low as 5% for very large switching systems (thousands of lines) up to 100% in nonswitching applications such as channel banks. Clearly, the average power dissipation in a typical switching system is below 35 mW which facilitates board packing density and cost of power considerations.

Table 6. Typical Power Dissipation Per Line Using 2914 Combochip

	Duty Cycle	Power Dissipation
Central Office	5%	12 mW
PABX	15%	25 mW
Peak Hour C.O.	50%	73 mW
Channel Bank	100%	140 mW

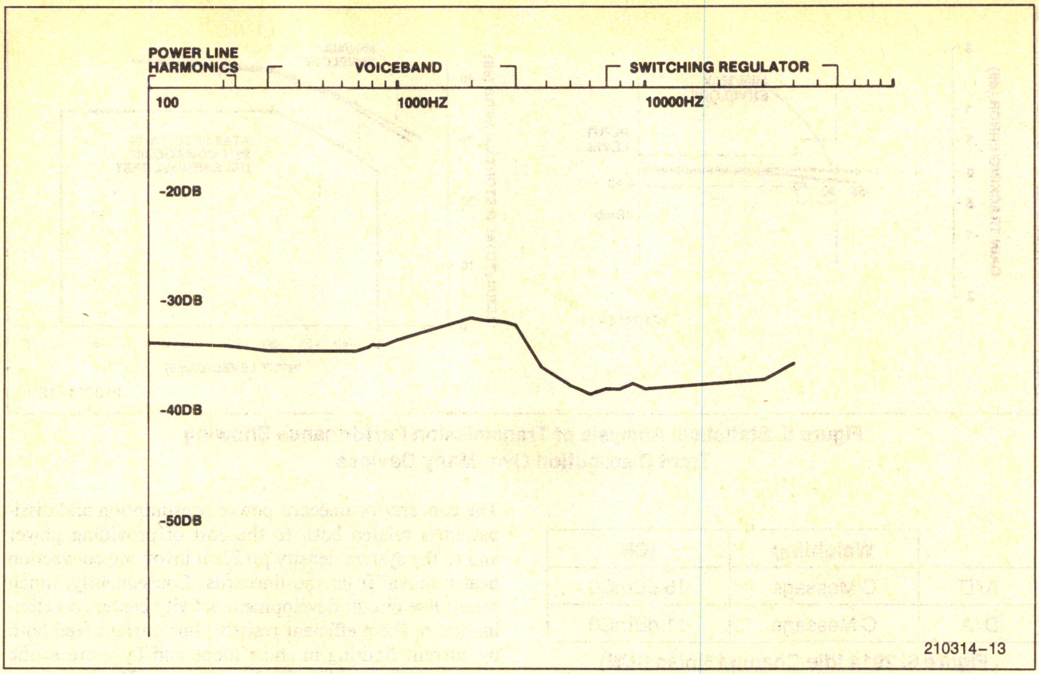


Figure 7. Wideband 2914 Power Supply Rejection Ratio (PSRR)

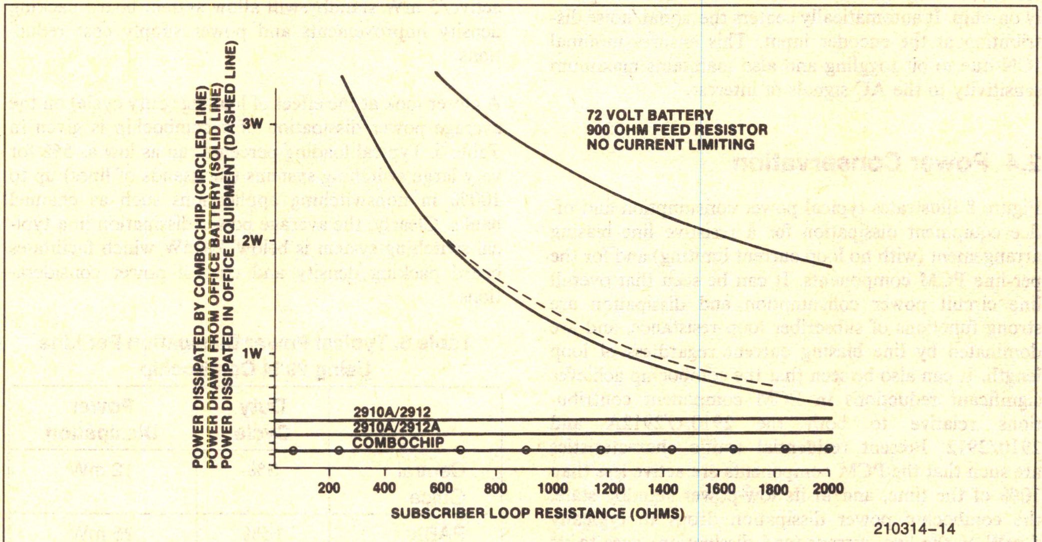
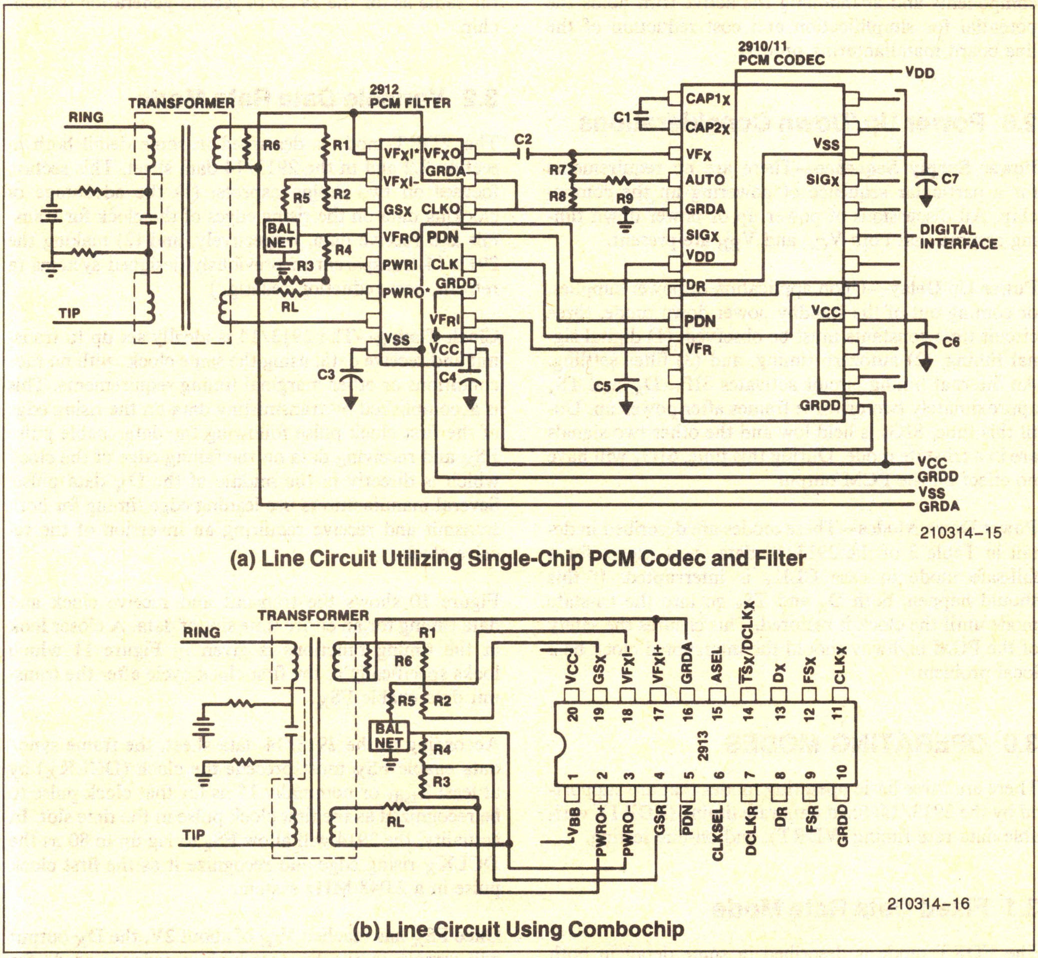


Figure 8. Line Circuit Power Consumption and Dissipation Curves

2.5 Elimination of Gain Trim in the Line Circuit

Four resistors—R1–R4 of Figure 9—on the transformer side of the PCM components are used to establish appropriate transmission levels at the PCM components and are, at first glance, equivalent in the two cases. However, a significant reduction in linecard manufacturing costs associated with individual line trim (or mop-up) is possible with the combochip. The need for this trim is dictated by system gain contrast specifications which typically require that the line-to-line gain variation shall not exceed 0.5 dB, which translates to 0.25 dB for each (transmit and receive) channel. Table 7 shows that the major portion of this gain variation has previously been in the nominal insertion loss of the PCM filter and in the uncertainty of the reference voltage of the codec. With this cumulative 0.15 dB uncertainty in the PCM components themselves, the system manufacturer had no choice but to resort to the cost and manufacturing complexity of the active trim. The combochip, however, can be trimmed during its manufacture to a nominal tolerance of ±0.04 dB which includes uncertainties in both the filter and codec voltage reference functions. This leaves 0.21 dB uncertainty to variations in the other line circuit elements and to temperature and supply variations.

The variation in combochip gain with supply and temperature has also been improved to allow as low as



(a) Line Circuit Utilizing Single-Chip PCM Codec and Filter

(b) Line Circuit Using Combochip

Figure 9. Schematics of the Codec/Filter Function and the 2/4 Wire Hybrid Transformers

Table 7. Gain Trim Budget for Codec/Filter Functions

Device	Manufacturing Uncertainty (Initial)	ΔT ΔSupplies	Total	Variation* Budget for Other Components
2910 2912	±0.1 ±0.05 ±0.15	±0.1 ±0.05 ±0.15	±0.3 dB	0 dB
2914	±0.04	±0.08	±0.12 dB	±0.13 dB

*Assumes 0.5 dB end to end gain contrast specifications.

0.08 dB variation over supplies and temperature so that more than half the system specification could be reserved for transformer, wiring, and resistor uncertainties. This possibility of using fixed precision gain trim components and abandoning the active trim holds the potential for simplification and cost reduction of the line board manufacturing process.

2.6 Power Up/Down Considerations

Power Supply Sequence—There are no requirements for a particular sequence of powering up the combochip. All discussions of power up or power down timing assume that both V_{CC} and V_{BB} are present.

Power Up Delay—Upon application of power supplies, or coming out of the standby power down mode, three circuit time constants must be observed: (1) digital signal timing, (2) autozero timing, and (3) filter settling. An internal timing circuit activates SIF_r, D_X, and TS_X approximately two or three frames after power up. Until this time, SIG_r is held low and the other two signals are in a tri-state mode. During this time, SIG_X will have no effect on the PCM output.

Power Down Modes—These modes are described in detail in Table 3 of the 2913/14 data sheet except for a fail-safe mode in case CLK_X is interrupted. If this should happen, both D_X and TS_X go into the tri-state mode until the clock is restored. This ensures the safety of the PCM highway should the interrupted clock be a local problem.

3.0 OPERATING MODES

There are three basic operating modes that are supported by the 2913/14: fixed data rate timing (FDRT), variable data rate timing (VDRT), and on-line testing.

3.1 Fixed Data Rate Mode

The FDRT mode is described in some detail in both section 2.2 of this note and in the 2913/14 data sheet. In addition, Intel Application Note AP-64 (Data Conversion, Switching, and Transmission using the Intel 2910A/2911A codec and 2912 PCM filter) also describes the basics of using the fixed data rate mode for first-generation codecs and filters which is essentially the same as for the 2913/14 second-generation combochip.

3.2 Variable Data Rate Mode

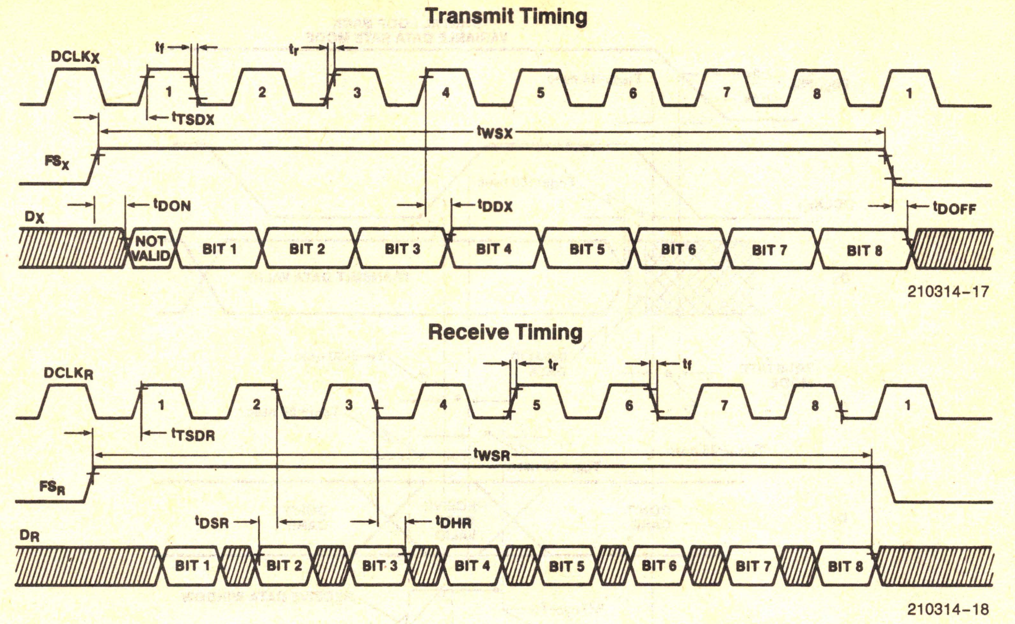
The VDRT mode is described in some detail both in section 2.2 and in the 2913/14 data sheet. This section focuses on two design aspects: (1) the advantage of clocking data on the rising edges of the clock for transmit and receive data, respectively, and (2) making the 2913/14 transparent in previously designed systems (a retrofit, cost reduction redesign).

Clock Timing—The 2913/14 is ideally set up to transmit and receive data, using the same clock, with no race conditions or other marginal timing requirements. This is accomplished by transmitting data on the rising edge of the first clock pulse following the data enable pulse FS_X and receiving data on the falling edge of the clock which is directly in the middle of the D_X data pulse. Several manufacturers use leading edge timing for both transmit and receive requiring an inversion of the receive clock.

Figure 10 shows the transmit and receive clock and data timing for an entire time slot of data. A closer look at the timing functions is given in Figure 11 which looks specifically at the first clock cycle after the transmit data enable FS_X.

According to the 2913/14 data sheet, the frame sync/data enable FS_X must precede the clock ($DCLK_X$) by at least T_{tsdx} or nominally 15 ns for that clock pulse to be recognized as the first clock pulse in the time slot. In actuality, the 2914 will allow FS_X to lag up to 80 ns the $DCLK_X$ rising edge and recognize it as the first clock pulse in a 2.048 MHz system.

Once FS_X has reached V_{IH} of about 2V, the D_X output will remain in the tri-state high-impedance mode for



NOTE:
All timing parameters referenced to V_{IH} and V_{IL} except t_{DON} and t_{OFF} which reference a high impedance state.

Figure 10. Variable Data Rate Timing for an Entire Time Slot

T_{don} or about 34 ns longer. It then comes out of tristate and will represent some data which is invalid until the valid data is available T_{DDX} or about 75 ns (100 ns worst case) after the clock rising edge. This means there is about 90 ns of invalid data after the tri-state mode. At this point there is valid data on the D_X highway that lasts for approximately one full clock cycle.

Since the D_X highway is tied directly to the D_r highway in digital loopback, the valid data above is now available to the receive channel with some propagation delay. The receiver is only interested in the data for about a 50 ns (110 ns worst case) window centered about the falling edge of the $DCLK_r$ clock which occurs about half a clock cycle from the FS_r rising edge. The window width is equal to the data set-up time. T_{dsr}, plus the clock fall time, T_f, plus the data hold time, T_{dhr}. Information at any other time on the D_r highway falls into the DON'T CARE category.

Retrofitting the 2913/14—Several switching/transmission systems have been designed using first-generation codecs which operate at data rates from 64 Kbps to 2.048 MBps. In addition, they may have been designed using the rising clock edges for both transmit and receive data.

Other aspects of these older designs could be relative skewing between the sync pulses (Data Enable) and the clock pulses in such a way that the sync pulse occurs after (Lags) the first clock pulse rising edge. All of these conditions can be easily handled using the variable data rate timing mode of the 2913/14 plus some simple external logic. By the addition of this logic, the 2913/14 becomes transparent to the older design thereby allowing an upgrade in performance while having no impact on backplane wiring or on system control hardware/software. In addition, many of the features of the 2913/14 may be incorporated, such as the test modes, which provide additional capabilities beyond those available in the original design and at a lower cost.

The circuit diagram in Figure 12 shows the maximum amount of additional random logic that could be necessary to make the 2913 or 2914 completely transparent at the linecard level (no impact on backplane wiring or timing). The inverter on $DCLK_R$ inverts all the receive clocks for each linecard. This inverter is only needed if (1) the transmit and receive clocks are inverted at the system/backplane level (as opposed to the linecard level) and (2) the previous design used only rising (or falling) edges to clock the transmit/receive data.

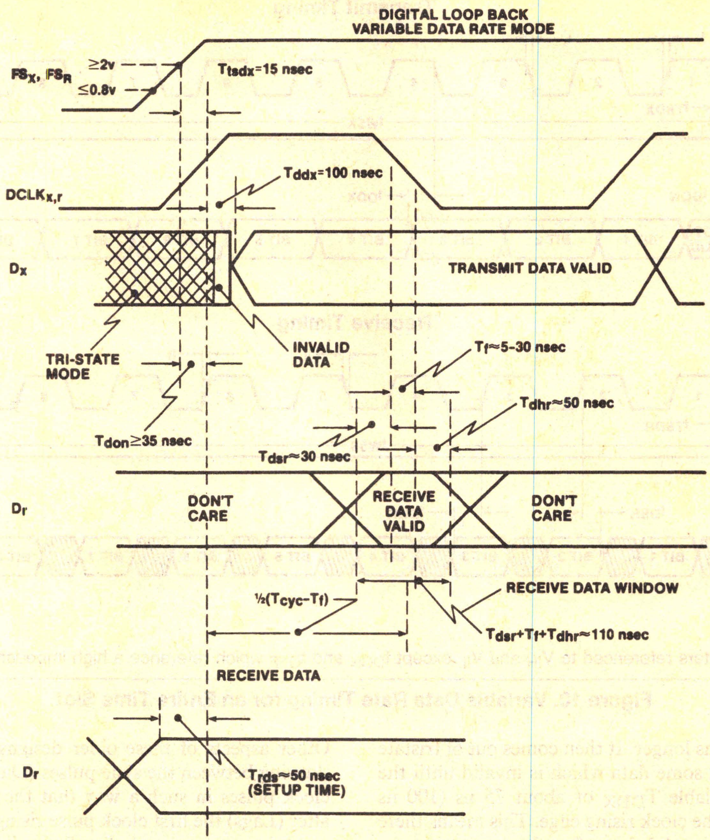


Figure 11. Waveform Timing Diagrams for the 2913/14

3.3 On-Line Test Modes

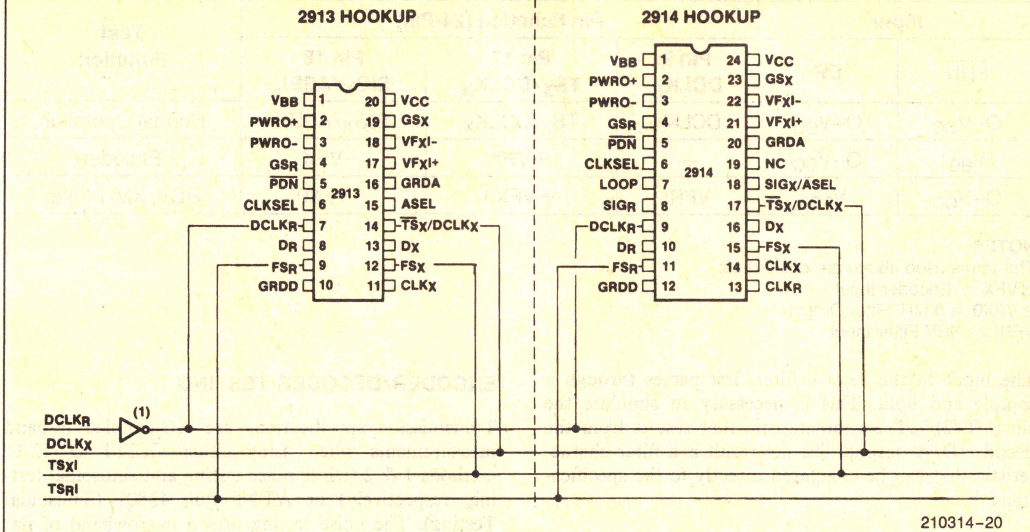
Two modes are available which permit maintenance checking of the linecard up to the SLIC/combochip interface, including the PCM highways and time slot interchanges. Tests include time slot-dependent error checking. The two test modes are called "redundancy testing" and "analog loopback." These test modes are described in detail in Section 4.3.

4.0 MULTIMODE TEST CAPABILITIES

The 2913/14 was designed with every phase of design, manufacturing, and operation taken into consideration. In particular, several test modes have been implemented within the device with essentially no increase in the package size or pin count. These test modes fall into three categories: design/prototype tests, manufacturing tests, and on-line operation tests; see Table 8.

4.1 Design/Prototype Testing

In the design of a linecard prototype or in the qualification of a device, it is often helpful to have direct access to the internal nodes at key points in the LSI system. Some manufacturers even dedicate pins specifically for this function. The Intel 2913/14 approach was to reduce cost by using multifunction pins and smaller packages to achieve this goal. Measurements through these multipurpose pins will typically yield full device capability against performance specifications, however *these measurements are not included in the device specifications.* This is done for two reasons: first, to save manufacturing cost by eliminating unnecessary tests and specifications, and, second, more cost effective manufacturing test techniques are available, as discussed in section 4.2.



NOTE:
(1) One inverter per linecard.

Figure 12. Circuit Diagram Showing Connections Needed to Retrofit the 2913/14 into Existing Variable Data Rate Systems

Table 8. Multimode Testing for Each Level from Design to On-Line Operation

- Design/Prototype Testing
  - Direct access to transmit codec inputs
  - Direct access to the receive filter input and the transmit filter differential outputs
- Manufacturing Tests
  - Standard half channel tests for combined codec/filters
  - Filter response half channel measurements
- Operation On-Line Tests
  - Analog loopback for testing PCM and codec analog highways
  - Redundancy checks with repeatable D_X outputs

Table 9 gives the input control pin values and the corresponding functions assigned to the key test pins on the 2914 for the design test modes.

Transmit Coded (Encoder)—The transmit filter can be bypassed by directly accessing the differential input of the transmit encoder with an analog differential drive signal. Table 9 shows the control pin voltages and the input pins for this test. This test mode permits DC testing of the encoder which is otherwise blocked by the AC coupling (low frequency reject filter) of the transmit filter.

Transmit and Receiver Filter—Table 9 shows the control values that permit access to the differential outputs of the transmit filter and the single-ended input to the receive filter. The voltage difference between the transmit filter outputs represents the filtered output that will be encoded. By driving VF_XI (single ended or differentially), the transmit filter response is obtained as a differential output. The final stage is the 60 Hz reject filter which is a switched capacitor filter sampled at an 8 KHz rate. When measured *digitally* (after the encoder), the filter characteristic is obtained directly: however, when measured in analog, a sin $(\omega T/2)/\omega T/2$ correction factor must be included.

Table 9. 2914 Test Functions and Control Inputs for the Design Test Modes

Input		Pin Function (24-Pin)			Test Function
$\overline{PDN}$	DR	Pin 9 $DCLK_R$	Pin 17 $TS_X/DCLK_X$	Pin 18 $SIG_X/ASEL$	
$0-V_{CC}$	$0-V_{CC}$	$DCLK_R$	$TS_X/DCLK_X$	$SIG_X/ASEL$	Normal Operation
V_{BB}	$0-V_{CC}$	—	+VFX	−VFX	Encoder
$0-V_{CC}$	V_{BB}	VFRI	+VFX0	−VFX0	RCV, XMIT Filter

NOTES:
The terms used above are defined as:
±VFX = Encoder Input
±VFX0 = XMIT Filter Output
VFRI = RCV Filter Input

The input to the receive filter first passes through a sample and hold. This is necessary to simulate the sin ($\omega T/2$)/$\omega T/2$ characteristic that results from the decoder D/A output. The net result is a filter characteristic that can be compared directly to the specifications.

Start-up Procedure for Test Modes—To place the 2913/14 in the test mode it is first necessary to operate the device for a few ms in normal operation. Then V_{BB} can be applied to the control pins to select the desired test access.

4.2 Production Testing

While it may be convenient for the designer to have access to both the filter and the codec inputs and outputs during the design or evaluation phase the final product will always use the filter and codec circuits together with all signals passing through both on the way to or from the PCM highways. It therefore makes sense to perform all manufacturing measurements with the device configured in its normal operating mode, i.e., all measurements should be complete filter/codec half channel measurements. This approach not only tests the combo as it will actually be used, but also saves time and money by eliminating separate measurements and correlation exercises to determine the full half channel performance.

Since the transmission specifications of S/D, gain tracking, and ICN all require measurements which are "in-band" or "filter independent," the codec functions can be easily tested using conventional half channel measurement equipment. The apparent difficulty arises in trying to fully measure the filter characteristics beyond the half sampling frequency of 4 KHz. In fact, this is not really a problem with today's computer-based testing plus an understanding of the sampled data process which is discussed under "Filter Testing".

ENCODER/DECODER TESTING

Transmission specifications are AC-coupled in-band measurements when using either CCITT G.712.11 methods 1 & 2 (white noise testing and sinusoidal testing, respectively) or AT&T Pub 43801 (Sinusoidal Testing). The noise testing uses a narrowband of flat noise from 300 to 500 Hz to drive the filter/codec (either in analog or the equivalent digital sequence for the transmit/receive channels, respectively). The resulting harmonic products are used to determine S/D. Likewise, gain tracking is also determined from this signal input. Sinusoidal testing uses a tone at 1.020 KHz for S/D measurements and gain tracking measurements. Idle channel noise measurements require the combined filter/codec since it has long been shown that separate measurements of filters and codecs are difficult to relate to the combined measurement (usually there is no specific relationship because of the non-linear properties of the encoder/decoder operations). Typically the frequency response of ICN measurements is primarily determined by the weighting filter (either C message or psophometric, which are both AC-coupled, bandpass type filters).

The conclusion is that combined filter/codec testing in no way limits the measurement of half channel transmission parameters of S/D, G.T., or ICN.

FILTER TESTING

Testing the filter response, of the transmit and receive channels presents two separate test situations which, in some ways, are mirror images of one another. With the transmit side, signals may be introduced at any frequency to test the filter response. At the output of the filter, the resulting signals are sampled at 8 KHz and digitized resulting in a sequence of PCM words representing the samples of filtered input signal. On the receive side, a digital PCM sequence of samples representing the driving signal is converted to an analog signal by the decoder and can be measured at the filter output in analog form.

Sampling Process—In both cases of testing the filter, the signal eventually is in a sampled form. Since the sampling rate is fixed at 8 KHz, all signals must be represented below 4 KHz (half the sampling frequency). This means that the PCM bit stream can only represent signals at frequencies below 4 KHz. If a signal above 4 KHz is sampled, those samples appear exactly as if the signal was at a frequency mirror imaged about 4 KHz. Two examples include signals at 5 KHz and 7 KHz which will result in samples that look like signals of 5−8 KHz = 3 KHz and 7−8 KHz = 1 KHz, respectively.

Conversely, the sampling process produces replicas (aliasing) of the sampled signal around multiples of the sampling frequency. Therefore, if two signals are introduced digitally representing 1 KHz and 2 KHz, there will also be frequency components located at 8 KHz = ±1 KHz and 8 KHz = ±2 KHz, and so on for all multiples of 8 KHz. Thus it is possible to generate frequencies at arbitrary values after sampling by controlling the frequency of each signal within the 4 KHz input band regardless of whether it is in analog or PCM.

When an analog signal is sampled, the frequency components generated are all of the same amplitude as the corresponding input spectral components. Therefore, on the transmit side, measurements made from the PCM data will have a throughput gain of unity except where components are superimposed (e.g., a 4 KHz input signal will have an alias component at 4 KHz which may double the amplitude at 4 KHz when the two components are combined).

When an analog signal is reconstructed from digital samples, it goes through a sample and hold stage which has the effect of imposing a weighting function on the resulting spectral components that is represented by

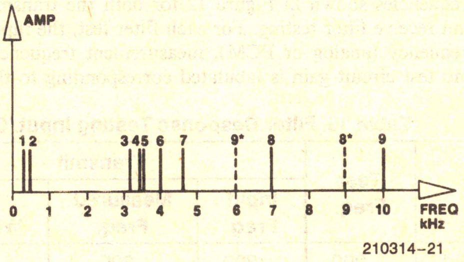
$$\mathrm{Sinc}\left[\frac{\omega T}{2}\right] = \frac{\mathrm{Sin}\left(\frac{\omega T}{2}\right)}{\frac{\omega T}{2}}$$

where ω is the actual spectral component frequency going into the filter, and T is the width of the hold pulse at the decoder output. For the 2913/14, the analog output is held the full sample period of 125 μs (1/8000 Hz) so that a frequency component at f_t will have a weighting of

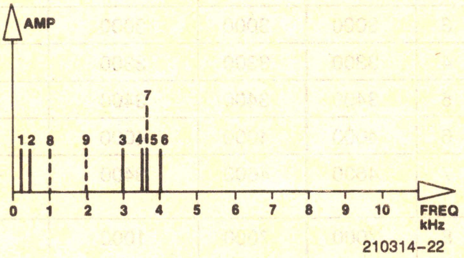
$$W = \left(\frac{8000}{\pi f_t}\right) \mathrm{Sin}\left[\frac{\pi f_t}{8000}\right]$$

Transmit Filter Test Approach—Two approaches can be used for half channel testing of the transmit filter characteristic: (1) input analog test frequencies and perform an FFT on the corresponding PCM samples that are generated to determine spectral frequencies and amplitudes at the codec output, or (2) use an "ideal" D/A converter on the PCM samples to convert the digital data back to analog so that the spectral amplitudes and frequencies can be determied using analog circuits such as spectrum analyzers or filter banks. In either case, the effects of sampling will be the same. Figure 13 shows two spectral diagrams of amplitude versus frequency. The top diagram represents the locations of nine test frequencies corresponding to the seven specified frequencies in the 2913/14 data sheet plus a component at 7 KHz and one at 10 KHz. The bottom figure shows the "equivalent" spectral component locations when carried in the PCM bit stream. As an example, frequency #8 is located at 7 KHz. The corresponding PCM frequency is seen in the lower figure at 1 KHz. Note also that the analog component at 9 KHz (see #8*) would also generate the 1 KHz component in the PCM data.

To test the filter, the desired test frequencies are introduced in analog to the filter input in such a way that there is no confusion as to where the resulting component will be after sampling (i.e., don't simultaneously put in 1 KHz and 7 KHz since both of these inputs result in a 1 KHz component in the PCM data). Then, using either technique (FFT or analog) mentioned above, measure the amplitude of the corresponding



(a) Analog Signal Frequency Locations



(b) PCM Representation of the Signals in (a)

Figure 13. Spectral Properties of the Filter Test Frequencies in Analog and PCM

sampled component. The difference between that amplitude and the input amplitude represents the filter attenuation *at the frequency of the input signal*. So, if the signal was at 7 KHz, the FFT will determine the amplitude of the corresponding 1 KHz signal. The amplitude change relative to the input will represent the filter attenuation at 7 KHz.

Receive Filter Test Approach—In this case, the PCM test signals can be generated directly from digital circuits or by going through an "ideal" A/D (companded) to generate the PCM samples. Since these samples represent frequencies below the half sampling rate, Figure 12(b) now represents the input signals and 12(a) the output, but with one significant difference—a Sinc[π f_I/8000] weighting function is imposed on all the frequency components because of the decoder sample and hold output. At the filter output, the spectral component amplitudes will include the effect of the filter response *and* the weighting function measured at the actual test frequency. The receive filter includes a compensation network for the weighting function in its passband. Therefore, inside the passband (300 Hz to 3.4 KHz) the measured amplitudes should be compared directly to the data sheet specifications. Frequencies outside the passband must be compensated for the weighting function first to determine the true filter response.

Summary of Filter Testing—Table 10 lists the nine test frequencies shown in Figure 12 for both the transmit and receive filter testing. For each filter test, the input frequency (analog or PCM), measurement frequency, and test circuit gain is tabulated corresponding to the desired test frequency. The various weighting values are easily handled by computer-based test equipment since the inverse weighting function can be stored in the computer and applied to each measured amplitude as appropriate.

4.3 Operational On-Line Testing

Two test modes are available which facilitate on-line testing to verify operation of both the combochip and the entire switching highway network. The first is simply the capability to duplicate the same D_X transmission in multiple PCM time slots (redundancy checking), and the second is the analog loopback capability which allows the testing of a call completion through the entire PCM voice path including the time slot interchange network.

Redundancy Checking—A feature of the 2913/14 is that the same 8-bit PCM word can be put on the D_X highway in multiple time slots simply by holding the frame sync/data enable (FS_X) high and continuing to supply clock pulses (CLK_X or $DCLK_X$). If the data enable was held high for multiple time slots, each time slot would have identical data in it. By routing this data through the PCM highways, time slot interchanges, etc., and then correlating the data between time slots, it would be possible to detect time slot-dependent data errors. When this test mode is used, no other data will be generated for the transmit highway until the frame sync returns low for at least one full clock cycle.

Table 10. Filter Response Testing Input/Output Frequencies and Amplitude Gain Schedule

	Test Freq.	Transmit			Receive		
		Input Freq.	Measured Freq.	Amp Weighting	Input Freq.	Measured Freq.	Amp Weighting
1	200	200	200	1	200	200	1
2	300	300	300	1	300	300	1
3	3000	3000	3000	1	3000	3000	1
4	3300	3300	3300	1	3300	3300	1
5	3400	3400	3400	1	3400	3400	1
6	4000	4000	4000	0 to 2	4000	4000	0 to 2
7	4600	4600	3400	1	3400	4600	$\mathrm{Sinc}\left[\frac{4600\,\pi}{8000}\right]$
8	7000	7000	1000	1	1000	7000	$\mathrm{Sinc}\left[\frac{7000\,\pi}{8000}\right]$
9	10000	10000	2000	1	2000	10000	$\mathrm{Sinc}\left[\frac{10000\,\pi}{8000}\right]$

Analog Loopback—The 2914 (2913 does not have this feature) has the capability to be remotely programmed to disconnect the outside telephone lines and tie the transmit input directly to the receive output to effect analog loopback within the combo chip. This is accomplished by setting the LOOP input to V_{CC} (TTL high). The result is to disconnect VF_XI+ and VF_XI- from the external circuitry and to connect internally PWRO+ to VF_XI+, GS_r to PWRO−, and VF_XI- to GS_X (see Figure 14).

With this test set up, the entire PCM and analog transmission path up to the SLIC can be tested remotely by assigning a PCM word to a time slot that is read by the combo being tested. This data is converted to analog and passed out of the receive channel. It is taken as input by the transmit channel where it is filtered and redigitized (encoded) back to PCM. The PCM word can now be put on the transmit highway and sent back to the remote test facility. By comparing the PCM data (individually or as a series of codes) the health of that particular connection can be verified.

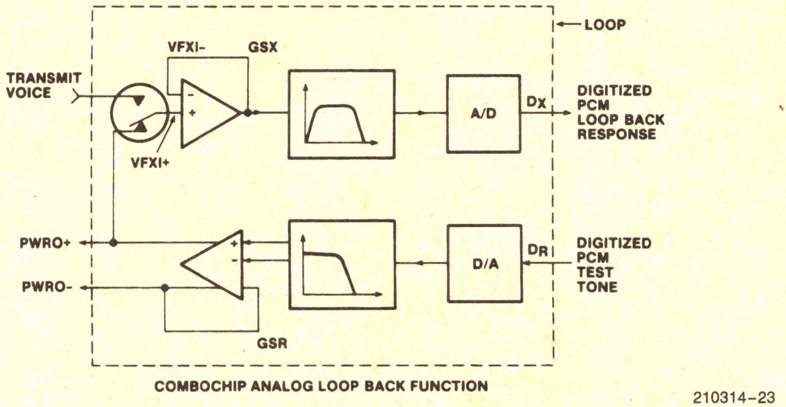


Figure 14. Simplified Block Diagram of 2914 Combochip in the Analog Loopback Configuration

Local Area Networking Boards and Software

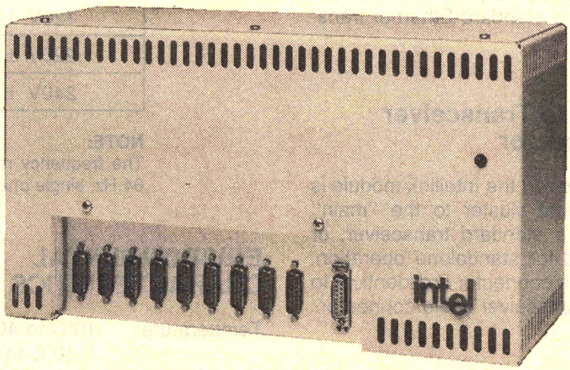
7

intel®

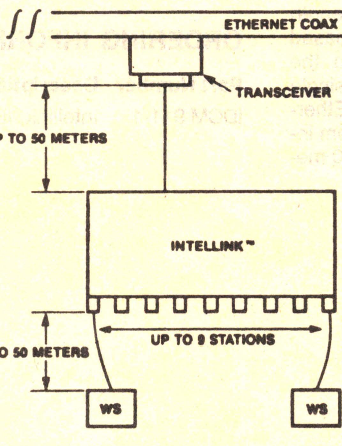
iDCM 911-2 INTELLINK™ ETHERNET* CLUSTER MODULE

- **Eliminates Need for Transceivers and Ethernet Coaxial Cable for a Local Cluster of Workstations**
- **Enables Local Cluster of Nine Workstations to Connect to Main Ethernet Cable with Only One Transceiver**
- **Permits Clustering of up to Nine Workstations in a Smaller Area**
- **Enables Workstations to be up to 100M from Main Ethernet Cable**
- **IEEE 802.3**

The Intellink™ Ethernet Cluster Module is a device used as a means of interconnecting up to nine Ethernet devices without the need for Ethernet coaxial cable and transceivers. The Intellink module forms a standalone Ethernet local area network with "interconnection" communication capability. The Intellink module (and attached devices) can optionally be connected to the Ethernet coaxial cable through a single transceiver.



210508-1



210508-2

Figure 1. Intellink™ Configuration

*Ethernet is a trademark of Xerox Corporation.

October 1986
Order Number: 210508-002

FUNCTIONAL DESCRIPTION

Intellink module performs the same functions as a standard Ethernet transceiver. It buffers receive and transmit data, detects attempts by two or more stations to gain access to the line simultaneously, signals the presence of a collision to the transmitting stations, and transmits the jam signal prior to initiation of the random back-off algorithm. It complies with all of the interface parameters set forth in IEEE 802.3 Specification.

Ethernet Work Station to Intellink™ Interface (WI) Connectors

There are nine WI interface connectors into which Ethernet-based systems can be connected. Each connector has the same signal pairs as does the equivalent connector on a standard Ethernet transceiver.

Intellink™ Module to Transceiver Interface (IT) Connector

The IT interface connector on the Intellink module is used to connect the local cluster to the "main" Ethernet cable through a standard transceiver, or can be left unconnected for standalone operation. The characteristics of this connector are identical to an Ethernet system to transceiver cable connector.

Topology

The Intellink module can function in standalone operation in which case it appears as a "zero length Ethernet segment" for up to nine Ethernet-based systems, or optionally can be connected to the "main" Ethernet coaxial cable through a single transceiver. When connected to the "main" Ethernet coaxial cable, it extends the Ethernet system interface to the transceiver from 50 meters to 100 meters. (Figure 1).

Physical Characteristics

Width: 14 in. (35.56 cm)
Height: 7.8 in. (19.81 cm)
Depth: 5.5 in. (13.97 cm)
Weight: 10 lb. (4.52 kg)

ELECTRICAL CHARACTERISTICS

Input Voltage Range

(Voltages AC RMS)

Voltage (15%)
100V ±15%
120V ±15%
220V ±15%
240V ±15%

NOTE:
The frequency range is 47 to 64 Hz, single phase.

ENVIRONMENTAL CHARACTERISTICS

Temperature:	10°C to 40°C Operating
	−40°C to 70°C Non-Operating
Humidity:	10% to 85% Operating
	5% to 95% Non-Operating

ORDERING INFORMATION

Part Number	Description
iDCM 911-1	Intellink, IEEE 802.3 compatible

intel®

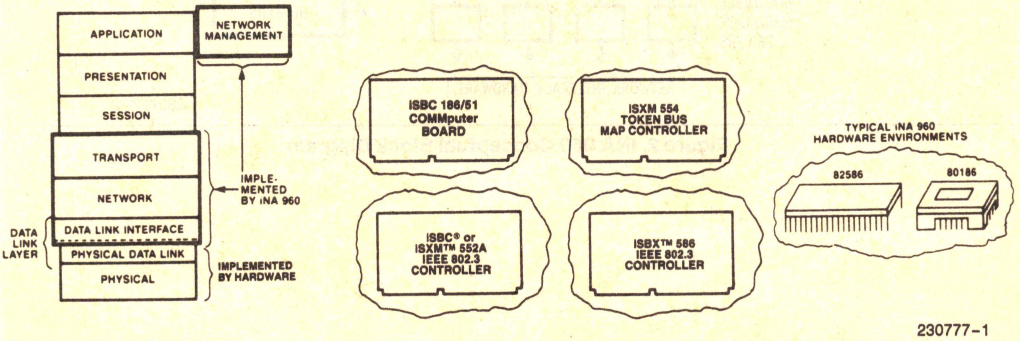
iNA 960/961 RELEASE 2.0 TRANSPORT AND NETWORK SOFTWARE MEMBER OF THE OpenNET™ PRODUCT FAMILY

- Certified ISO/MAP Standard Transport and Network Layer Software
- ISO 8073 Transport Class 4 Services
  - Multiple Virtual Circuit Connection Capability
  - Guaranteed Message Integrity
  - Data Rate Matching (Flow Control)
  - Variable Length Messages
  - Expedited Delivery
- ISO 8473 Network Class 3 Services
  - Connectionless Internetworking Capability
  - Supports End-Node Systems
  - Supports Internetwork Routers
- Highly Configurable for Multiple System Environments
  - As an iRMX® 86 Job
  - As a Stand-Alone Communications Processor System
  - Supports Other Host Operating System Independent Designs
- Connectionless Transport (Datagram) Services
- Data Link Drivers Support Many Hardware Environments
  - IEEE 802.3 Hardware Such as the iSBC® 186/51, iSXM™ 552(A), and iSBX™ 586 Boards and Various Designs Based on the 8086, 8088, or 80186 Processors and the 82586 LAN Coprocessor
  - IEEE 802.4 Hardware such as the iSBC® 554 Board
  - Others Definable by the User
- Comprehensive Network Management Services
  - Collection of Network Usage Statistics
  - Setting and Inspecting of Transport and Data Link Parameters
  - Fault Isolation and Detection
  - Boot Server

iNA 960 is a complete transport and network software system plus a comprehensive set of network management functions, data link drivers, and system environment features. It is highly configurable to allow optimized selection of features, parameters, data link drivers, and memory buffers for a variety of system environments.

iNA 961 is derived from iNA 960. It consists of preconfigured subsets of iNA 960 that are designed to operate with several specific COMMengine hardware environments. iNA 961 contains preconfigured load files ready for download to the hardware. Load files are included to support the iSXM 552 and iSXM 552A IEEE 802.3 COMMengines, and the iSBC 554 IEEE 802.4 (MAP) COMMengine.

iNA 960/961 is a mature, flexible, and ready-to-use software building block for OEM suppliers of networked systems for both technical and commercial applications. Using the iNA 960 software the OEM can minimize development cost and time while achieving compatibility with a growing number of equipment suppliers adopting the ISO and IEEE standards.



230777-1

September 1986
Order Number: 230777-003

FUNCTIONAL OVERVIEW

Using the ISO seven layer model for network communications, iNA 960 provides the services of layers four and three, the transport and network layers. The iNA 960 design is an implementation of the Class 4 services of the ISO standard 8073 connection oriented transport protocol. The iNA 960 transport layer provides a reliable full-duplex message delivery service on top of the internetworking capability offered by the network layer. The iNA 960 network layer is an implementation of the Class 3 services of the ISO standard 8473 connectionless network protocol. The network layer allows routing of information packets between different networks (each network is called a subnetwork). The network layer directs information packets to the packet delivery services of the IEEE 802.3 or IEEE 802.4 data link and physical layer functions.

Consisting of linkable object modules, the iNA 960 software can be configured to implement a range of capabilities and interface protocols. In addition to reliable process-to-process message delivery services, iNA 960 includes a datagram service, internetworking end-node capabilities, internetworking router node capabilities, boot server capabilities, a direct user access to the data link layer, and a comprehensive network management facility.

iNA 960 also contains a variety of client program interfaces, data link drivers, and a stand-alone operating system executive. As a result, iNA 960 is highly configurable to run under the iRMX 86 operating system, to run under its own operating system executive on an Intel iSXM IEEE 802.3 or IEEE 802.4 network board, or to run on a custom designed controller with an 8086, 8088, or 80186 processor coupled with an 82586 data link coprocessor.

The iNA 960 software also includes a comprehensive network management service. This facility enables the user to monitor and adjust the network's operation in order to optimize its performance.

For a conceptual block diagram of iNA 960, refer to Figure 2.

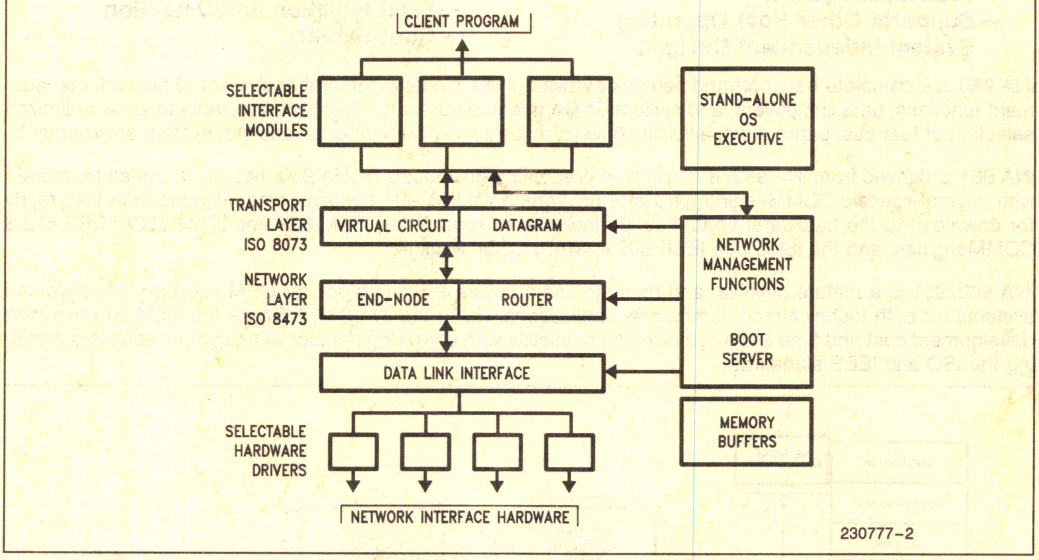


Figure 2. iNA 960 Conceptual Block Diagram

TRANSPORT LAYER

The Transport Layer provides message delivery services between client processes running on computers (network "nodes") anywhere in the network. Communicating client processes within the network are identified by a transport address that is a combination of a network address defining the network node and a transport service access point defining the interface point through which the client accesses the transport services. The transport address is supplied by the iNA 960 user for both the local and the remote client processes that are to be connected.

The iNA 960 transport layer implements two kinds of message delivery services: virtual circuit and datagram. The virtual circuit services provide a reliable point-to-point message delivery capability that ensures maximum data integrity and is fully compatible with the ISO 8073 Class 4 protocol standard. The datagram service provides a best-effort message delivery between client processes requiring less overhead and therefore allows higher throughput than virtual circuits.

Both the datagram and the virtual circuit services are optional and can be included when configuring iNA 960.

Virtual Circuit Services

Reliable Delivery: Data is delivered to the destination in the exact order it was sent by the source with no errors, duplications or losses, regardless of the quality of service available from the underlying network service.

Data Rate Matching (flow control): The Transport Layer attempts to maximize throughput while conserving communication subsystem resources by controlling the rate at which messages are sent. That rate is based on the availability of receive buffers at the destination and its own resources.

Multiple Connection Capability (Process Multiplexing): Several Processes can be simultaneously using the Transport Layer with no risk that progress or lack of progress by one process will interfere with others.

Variable Length Messages: The client software can submit arbitrarily short or long messages for transmittal without regard for the minimum or maximum network service data unit (NSDU) lengths supported by the underlying network services.

Expedited Delivery (optional): With this service the client can transmit up to 16 bytes of urgent data bypassing the normal flow control. The expedited data is guaranteed to arrive before any normal data submitted afterward.

Connectionless Transport (Datagram) Service

The datagram service transfers data between client processes without establishing a virtual circuit connection. The service is a "best effort" capability and data may be lost or misordered. Data can be transferred at one time to a single destination or to several destinations (multicast). The iNA 960 datagram service conforms to the ISO draft standard DIS 8602.

NETWORK LAYER

The network layer of iNA 960 provides the Class 3 connectionless network services specified by the ISO standard 8473 protocols.

The iNA 960 network layer provides the capability of connecting multiple different networks (called subnetworks) together and having information packets from one subnetwork routed to a destination on any other subnetwork. The network layer thus provides for two major capabilities:

- Internetworking
- Multiple subnets attached to one node

The iNA 960 network layer allows the user a variety of configurations. A node can be configured as:

- An internet end node belonging to a single subnetwork which is in turn connected to other subnetworks. In this configuration, the end node has the capability to address other nodes anywhere on the entire system of subnetworks.
- An internetwork router belonging to two or more subnetworks. In this case, only the ISO 8473 standard connectionless internetworking layer is configured on the node. The user can select the addressing and routing algorithms to be used. The iNA 960 network layer provides a routing algorithm with user changeable routing tables. The network layer also permits the future addition of address passing and routing algorithms as standards emerge. A router node can be configured with a variety of subnet data link and physical layers of mixed media types. The transport layer and above are not needed by this node.

- A multi-homed end system which is connected to two or more subnets. The network layer can provide routing between these subnets. In this case the transport layer is included and applications can run on this system and communicate on all subnetworks connected to it.
- A single network end node which can address nodes on one subnetwork only. This gives iNA 960 the transport layer functionality and a null network layer. The program interface to iNA 960 can be set up to accept the network address format of the ISO 8473 standard or of the previous draft ISO standard.

Data Link Drivers

The iNA 960 network layer has a variety of data link drivers for both IEEE 802.3 and IEEE 802.4 data link and physical layers. Specifically, IEEE 802.3 hardware drivers are included for the iSBC 186/51 COMMputer, the iSBC/iSXM 552 and 552A COMMengines, the iSBX 586 module, and 82586-based custom designed systems. An IEEE 802.4 data link driver is also included for the iSBC 554 token-bus MAP board. In addition, a user can add up to two user written subnetworks (when operating under the iRMX 86 operating system). Communication between the subnetwork drivers and the network layer is via request blocks and is based on the programmatic interface specified by iNA 960.

Router Capabilities

Since iNA 960 includes a wide variety of data link drivers and a flexible internetworking capability, a wide variety of internetworking configurations are supportable.

- With the iSBC 554 board and the iSBX 586 module, an IEEE 802.4 MAP token-bus to IEEE 802.3 Ethernet CSMA/CD router is supported. iNA 961 includes a preconfigured load file for this hardware configuration.
- With the iSBC 186/51 board and the iSBX 586 module, an IEEE 802.3 to IEEE 802.3 router is supported.
- With the iSBC 186/51 board (which has both an IEEE 802.3 port and a serial port), the user can link a separate serial data link driver (such as for X.25) to the iNA 960 network layer and produce an IEEE 802.3 to serial link router.

For full internetworking configurations, the user can set up the routing tables which are used for routing information packets between subnetworks. The routing tables can be changed during operation via a routing management facility. Information packets follow the routing path fixed by the routing table information.

NETWORK MANAGEMENT FACILITY

The Network Management Facility provides the user of iNA 960 with planning, operation, maintenance, and initialization services described below:

- Planning: This service captures network usage statistics on the various layers to observe network traffic and to help plan network expansion. Statistics are maintained by the layers themselves and are made available to users via a program interface with the NMF.
- Operation: This service allows the user to monitor network functions and to inspect and adjust network parameters. The goal is to provide the tools for performance optimization on the network.
- Maintenance: This service deals with detecting, isolating, and correcting network faults. It also provides the capability to determine the presence of other nodes on the network and the viability of their connection to the network.
- Initialization: NMF provides initialization and remote loading facilities for remote nodes on the network.

Network management provides distributed management of the network. The user can request any of the services to be performed on a remote as well as a local node. The NMF interfaces to every other network layer both to utilize their services and to access their internal data bases.

In support of the above services, the NMF capabilities include layer management, echo testing, limited debugging facilities, and the ability to down line load and dump a remote system.

The NMF software provides a routing management facility which can be used to change the internetworking routing tables. The routing tables are used by the network layer to route information packets between subnetworks.

The NMF provides the hooks for MAP-NET software (which provides layers 5 through 7 support for MAP networks) to support the network management functions in the MAP 2.1 specification. Thus, the MAP-NET user has a choice of selecting the Intel NMF network management functions or the MAP network management functions.

Layer management deals with manipulating the internal database of a layer. The elements of these data bases are termed objects. Some examples for objects are the number of collisions, the retransmission timeout limit, the number of packets sent, and the list of nodes to boot. NMF can examine and modify objects in a layer's data base.

An echo facility is provided. Using this facility, one node can determine if another node is present on the network or not, test the communication path to that node, and determine whether the remote node is functional.

NMF enables the user to read or write memory in any node present on the network. This feature is provided as an aid to debugging.

NMF can down line load any system present on the network. A simple Data Link protocol is used to ensure reliability. This facility can be used to load databases, to boot systems without local mass storage, or to boot a set of nodes remotely, thus ensuring that they have the same version of software, etc.

Dumping is an operation equivalent to memory read from the user's standpoint. However, dumping uses the Data Link facilities while memory read uses the transport facilities.

EXTERNAL DATA LINK (EDL)

The External Data Link option allows the user to access the Data Link Layer directly instead of having to go through the network and transport layers. This flexibility is useful when the user needs custom higher layer software or does not need the Network Layer and Transport Layer services (e.g. when sending "best effort" messages or running customer diagnostics).

Through the EDL, the capabilities supporting the lower layers in iNA 960 are made directly available to the user. EDL enables the user to establish and delete data link connections, transmit packets to individual and multiple receivers, and configure the data link software to meet the requirements of the given network environment.

SYSTEM ENVIRONMENT

iNA 960 is designed to run on hardware based on the 8086, 8088, or 80186 microprocessors and the 82586 LAN Coprocessor. The software can be configured to run under the iRMX 86 operating system or on a dedicated 8086, 8088, or 80186 processor separately from the host. The following section describes these two operating environments.

iRMX® 86 Operating System Environment

In this configuration, both the user program and iNA 960 are running under the iRMX 86 operating system. The communications software is implemented as an iRMX 86 job requiring only the iRMX nucleus for most operations. The only exception is the boot server option which also needs the iRMX 86 Basic IO System. The iSBX 586 IEEE 802.3 module is supported when iNA 960 runs under the iRMX 86 operating system. Also, the two user defined data link drivers are supported when iNA 960 runs under the iRMX 86 operating system. Figures 3 and 4 show two example hardware configurations supported by iNA 960 running under the iRMX 86 operating system.

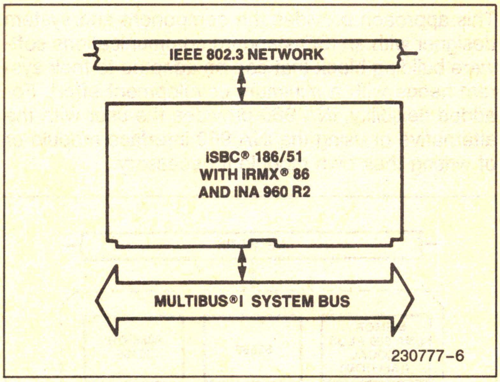


Figure 3. Configuration using iSBC® 186/51, iRMX® 86 and iNA 960

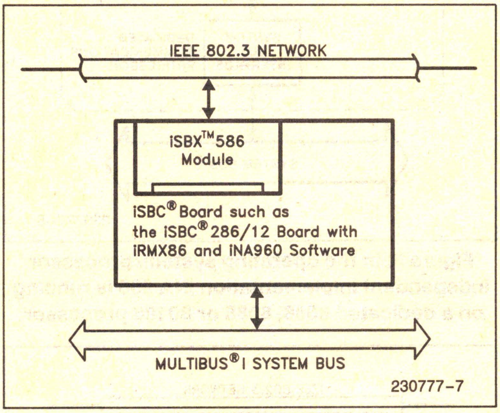


Figure 4. Configuration using an iSBC® Board and iSBX™ 586 Controller Module

Operating System Processor Independent Implementation

iNA 960 is also capable of operating in a stand-alone system environment under its own operating system executive. This mode of operation is appropriate in those systems where the iRMX operating system is not the primary operating system, where off-loading the host of the communications tasks is necessary for performance reasons, or where a cus-

tom designed communications front end processor configuration is being used. iNA 960 can be configured to support such implementations by providing network services on an 8086, 8088, or 80186 processor that in turn controls an 82586 LAN coprocessor. Figure 5 depicts the conceptual block diagram of this configuration. The iSBC/iSXM 552, the iSBC/iSXM 552A, and the iSBC 554 boards are MULTIBUS® I implementations of this architecture. Figures 6 and 7 depict examples of these implementations.

This approach provides the component and system designer with an ISO standard communications software building block that can be adapted to their system needs with a minimum development effort. For added flexibility, iNA 960 provides the user with the alternative of using the iNA 960 interface module or of writing their own module if necessary.

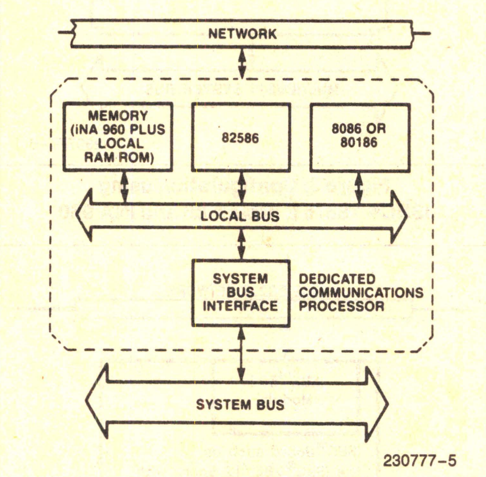


Figure 5. In the operating system/processor independent implementation iNA 960 is running on a dedicated 8086, 8088 or 80186 processor.

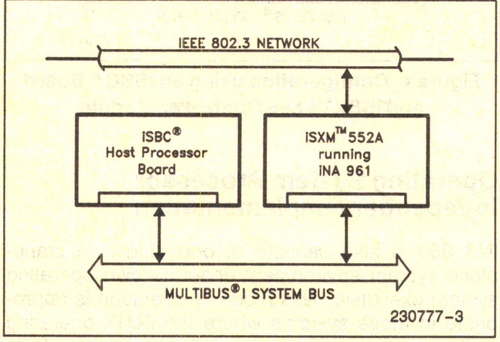


Figure 6. Configuration using the iSXM™ 552A and iNA 961

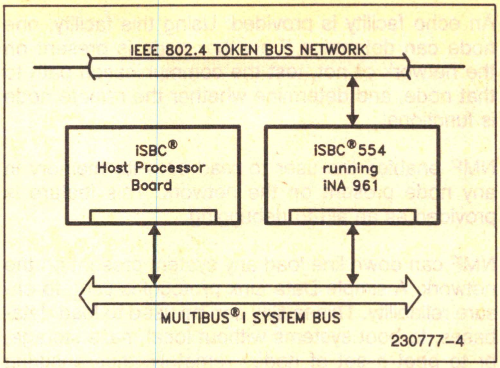


Figure 7. Configuration using the iSBC® 554 and iNA 961

USER INTERFACE

iNA 960 is designed to run both under the iRMX 86 operating system or on a dedicated communications front end processor separate from the host. In both environments, the user interface is based on exchanging memory segments called request blocks between iNA 960 and the client. The format and contents of the request blocks remain the same in both configurations with only the request block delivery mechanism changing.

Request blocks are memory segments containing the data to be passed from the user to iNA 960 (commands) or from iNA 960 to the user (responses). The iNA 960 request blocks consist of fixed format fields identical across all user commands and argument fields unique to the individual commands. Refer to Figure 8 for the standard request block format.

Issuing an iNA 960 command consists of filling in the request block fields and transferring the block to iNA 960 for execution. After processing the command, iNA 960 returns the request block with one of the pre-defined response codes placed in the response code field of the request block. The response code indicates whether the command was executed successfully or whether an error occurred. By examining the response code, the user can take appropriate action for that command.

The request block delivery mechanism is the means by which the host processor and the communications processor running iNA 960 software exchange the request blocks. iNA 960 provides three such mechanisms: the MIP (Multibus Inter-process Protocol), the BCB (Base Control Block), and a user-defined mechanism. The MIP interface is included for use in systems already supporting this protocol, the BCB is a simple interface for single host environments, and the user-defined interface accommodates unique application requirements.

FIELDS	WORD/BYTE	
Reserved (2)	WORD	FIXED FORMAT FIELDS (same for all commands)
Length	BYTE	
User I.D.	WORD	
Response Port	BYTE	
Return Mailbox Token	WORD	
Segment Token	WORD	
Subsystem	BYTE	
Opcode	BYTE	
Response Code	WORD	
Arguments	BYTE	ARGUMENTS (changes by command)
•	•	
•	•	
•	•	

Figure 8. iNA 960 Request Block Format

Transport Layer User Interface

The following table summarizes the user commands and the corresponding transport layer responses:

Command	Function
OPEN	Allocates memory for the connection database of a virtual circuit for connection to be established. The connection database contains data concerning the connection.
SEND CONNECT REQUEST	Requests connection to a fully specified remote transport address using specified ISO connection negotiation options.
AWAIT CONNECT REQUEST TRAN	Indicates that the transport client is willing to consider incoming connection requests based on pre-established acceptance criteria.
AWAIT CONNECT REQUEST USER	Indicates that the transport client is willing to consider incoming connection requests if the request meets the address and negotiation option criteria it passed to the client for further consideration.
ACCEPT CONNECT REQUEST	Indicates that the connection requested by a remote transport service is accepted by the client.
SEND DATA or SEND EOM DATA	With this command the client requests the transmission of the data in the buffers using the normal delivery service of the specified connection.
RECEIVE DATA	Posts normal receive data buffers for a specific connection or for a buffer pool used by a class of connections.
WITHDRAW RECEIVE BUFFER	Returns a previously posted receive buffer for use.
SEND EXPEDITED DATA	Transmits up to 16 bytes of data using the expedited delivery service. The expedited data is guaranteed to arrive at the destination before any normal data submitted afterward.
RECEIVE EXPEDITED DATA	Posts receive data buffers for expedited delivery for a specific connection or for a pool of buffers used by a class of connections.

Transport Layer User Interface (Continued)

Command	Function
WITHDRAW EXPEDITED BUFFER	Returns a previously posted expedited delivery receive buffer for use.
CLOSE	Terminates an existing connection or rejects an incoming connection request. Any normal or expedited data queued up to be sent will not be sent.
AWAIT CLOSE	Requests notification from the client of the termination of a specified connection.
STATUS	Returns status of the transport service connections.
SEND DATAGRAM	Requests transmission of the data in the buffers using the transport datagram service.
RECEIVE DATAGRAM	Posts a receive buffer for a specific receiver or a class of receivers to receive data from a transport datagram.
WITHDRAW DATAGRAM BUFFER	Returns a previously posted datagram buffer for use.
ADD DATAGRAM MULTICAST ID	Allows a client to belong to a group and receive datagrams sent to this group in addition to receiving datagrams specifically addressed to the client.
DELETE DATAGRAM	Allows a client to remove themselves from a multicast group.

Network Management Layer User Interface

Command	Function
READ OBJECT	Returns the value of the specified object to the client.
SET OBJECT	Sets the value of an object as specified by the client.
READ AND CLEAR OBJECT	Returns the value of the specified object to the client then clears the object.
ECHO	This function is used to determine the presence of a node to test the communication path to that node and to ascertain the viability and functionality of the remote host addressed.
UP LINE DUMP	Requests a remote node to dump a specified memory area.
READ MEMORY	Reads memory of the specified network node.
SET MEMORY	Sets memory of the specified network node.
FORCE LOAD	Causes a node to attempt a remote load from another node.

External Data Link Interface

Command	Function
CONNECT	With this command the client establishes a data link connection.
DISCONNECT	Eliminates a previously established connection.
TRANSMIT	Transmits data contained in buffers specified by the client.
POST RECEIVE PACKET DESCRIPTOR	Allocates memory for maintaining records on receive data buffers. Also may be used to allocate memory for buffering receive data.
ADD MULTICAST	Adds an address to the list of data link multicast addresses.
REMOVE MULTICAST ADDRESS	Removes an address from the list of data link multicast addresses.
SET DATA LINK ID	Sets up a unique data link ID for the node.

CONFIGURING iNA 960

iNA 961 contains preconfigured subsets of iNA 960 that are designed to execute on specific hardware configurations such as the iSXM 552A and the iSBC 554 boards. The preconfigured load files in iNA 961 are ready for downloading to the hardware and therefore require no software configuration effort by the customer.

iNA 960 is highly configurable for a variety of system environments, and it therefore allows configuration and optimization by the customer. iNA 960 is configurable at the object code level.

In order to adapt iNA 960 to a specific system environment, the user must configure the software to define the desired functions, to select the appropriate user interface, to set the layer parameters, and to set up for the specific hardware configuration.

There are a number of capability combinations the user may elect to implement in their application. At the transport layer level, the options are virtual circuit service with or without expedited delivery, or datagram service, or both. At the network layer level, the options are to use the ISO 8473 internet layer or to use a null network layer. At the data link level, the user may include or exclude the External Data Link interface.

The Network Management Facility is also optional. When it is configured in, the user may also include the boot server module. These capabilities can be made available simply by linking in the corresponding software modules. The interface options are also implemented in a modular fashion. The user links in the desired module to set up for the iRMX 86 operating system or the operating system independent configurations.

Layer parameters and configuration options are first edited into layer configuration files, then assembled and linked into iNA 960. Layer parameters adjust the network's operation to match the usage pattern and the available resources. For example, within the Transport Layer, the flow control parameters, the retransmission timer parameters, the transport data base parameters, etc. can be set via this process.

During the configuration process, the user also sets up for the required hardware configuration, such as port addresses, interrupt levels, number of memory buffers, etc. For the flow diagram of configuring iNA 960, refer to Figure 9.

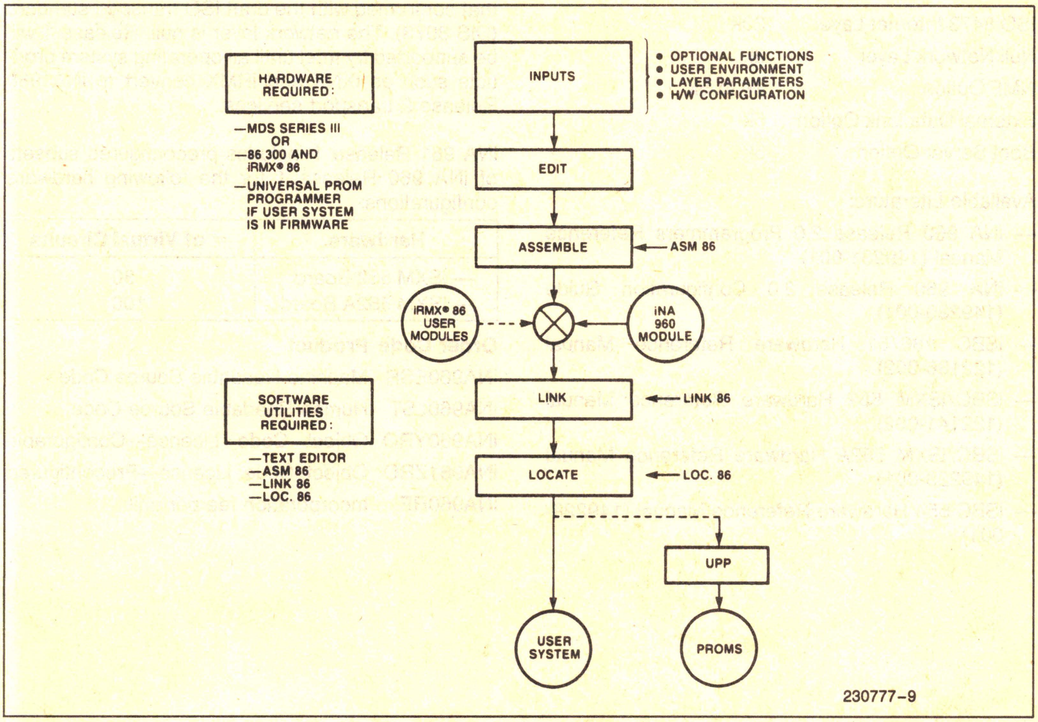


Figure 9. The Configuration Process for iNA 960

SPECIFICATIONS

Hardware Supported

— iSBC 186/51 Communication Computer

— iSBC/iSXM 552 and 552A Ethernet COMMengines

— iSBC 554 Token Bus (MAP) COMMengine

— iSBX 586 Ethernet Data Link Engine when configured with a supporting iSBC or iSXM board.

Typical Throughput at Transport

Environments:

186/51 and iRMX 86 Operations System	50k to 200k bytes/sec
Dedicated 80186/82586 COMMengine	100k to 300k bytes/sec

Memory Requirements (in bytes):

Base System	12k plus configurable buffer memory
Normal Virtual Circuit Option	18k plus configurable buffer memory
Expedited Delivery Option	2k
Datagram Option	3k plus data base memory
ISO 8473 Internet Layer	20k
Null Network Layer	2k
NMF Option	1k to 5k
External Data Link Option	5k
Boot Server Option	5k

Available Literature:

— iNA 960 Release 2.0 Programmers Reference Manual (149231-001)

— iNA 960 Release 2.0 Configuration Guide (149230-001)

— iSBC 186/51 Hardware Reference Manual (122136-002)

— iSBC/iSXM 552 Hardware Reference Manual (122141-002)

— iSBC/iSXM 552A Hardware Reference Manual (149228-001)

— iSBC 554 Hardware Reference Manual (149229-001)

— MAP—NET Programmers Reference Manual (149227-001)

— RMX—NET Programmers Reference Manual (122323-002)

ORDERING INFORMATION

iNA 960 is the order code for the fully configurable version of iNA 960 with the full ISO standard transport and network services. Licenses are available for both the object and the source code.

iNA 961 is the order code for a preconfigured version of iNA 960 for the following hardware configurations:

Hardware	Network Layer	# of Virtual Circuits
— iSXM 552 Board	null	30
— iSXM 552A Board	null	100
— iSXM 552A Board	internet	100
— iSBC 554 Board	internet	100
— iSBC 554/iSBX 586 Boards	internet (router)	(no transport)

iNA 960 release 1 is the former version (available since 1984) of fully configurable iNA 960 software that conformed with the draft ISO transport standard (DIS 8073). The network layer is null. Release 1 will be supported by Intel until all operating system products such as iXNX and iRMX convert to iNA 960 Release 2 transport services.

iNA 961 Release 1 includes preconfigured subsets of iNA 960 Release 1 for the following hardware configurations:

Hardware	# of Virtual Circuits
— iSXM 552 Board	30
— iSXM 552A Board	100

Order Code	Product
iNA960ESR	Machine Readable Source Code
iNA960LST	Human Readable Source Code
iNA960YRO	Object Code License—Configurable
iNA961ZRO	Object Code License—Preconfigured
iNA960RF	Incorporation fee per unit.

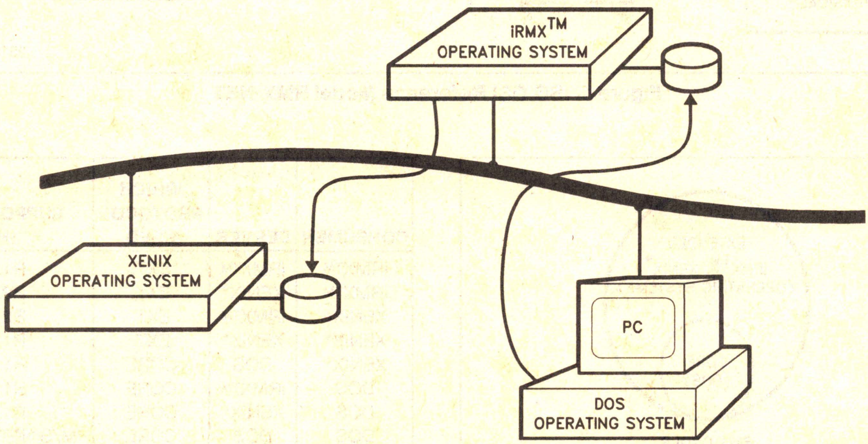
iRMX™ NETWORKING SOFTWARE

MEMBER OF THE OpenNET™ PRODUCT FAMILY

- Transparent Network File Access
  - Remote files can be worked with as if they were local
- Connects iRMX™, XENIX* and DOS systems on the LAN**
  - Compatible with XENIX Networking Software (XENIX* NET) and MS-NET/IBM PC Networking program
- Runs under iRMX™ 86 Operating System
- Existing applications can be distributed without change
- Supports OpenNET™—Ethernet hardware and software
  - iSXM™ 552 Transport Engine
  - iSBC® 552 COMMengine
  - iSBC® 186/51 COMMputer™
  - iNA 960 Transport software
- Supports file server applications
  - Based on iRMX™ 86 Basic I/O system
- Distributed name server

The Intel OpenNET™ iRMX™ Network File access software provides transparent file access between iRMX and XENIX* and iRMX and MS/DOS systems across a LAN. Users can use local file systems commands to read, write, open, close, etc. files residing at remote iRMX, MS/ or PC/DOS and XENIX systems. IRMX NET implements the upper layer ISO OSI protocols used by the IBM PC Network Program and XENIX NET. Interoperation among these systems is supported by Intel's LAN product line including the iSXM 552 Transport engine, the iSBC® 552 COMMengine, the iSBC® 186/51 COMMputer™ and the iNA 960 Transport software. Networked iRMX systems serve in a wide range of applications including real time transactions, automated testing, data collection, communications switching, etc.

*XENIX is a trademark of Microsoft Corp.
**RMX to XENIX interoperation will be fully qualified only in R2.0 and up.



231372-1

iRMX™-NET FUNCTIONAL DESCRIPTION

iRMX™-NET provides transparent remote file access capability through a file consumer and a file server module. The consumer intercepts file commands from the local user and transmits them across the LAN to the server at the node where the target file resides. The server receives, interprets and executes the command acting as a user to its local file system. The user has the option of configuring either or both in his target system.

RMX-NET also includes a name server which provides name-to-address mapping. The iRMX-NET file consumer uses the name server to find the physical address of the referenced system.

The capabilities allow iRMX systems to interoperate over the LAN with XENIX systems configured with XENIX-NET or DOS systems using MS-NET or IBM PC Network Program. This interoperation entails accessing data and loading programs through the network, sharing common servers and communication between users.

The network file service requires the support of an underlying ISO 8073 compatible transport service provided by the iNA 960 network software running on the iSBC 186/51 COMMputer or the iSXM 552/iSBC 552 boards. In terms of the ISO OSI reference model iRMX-NET, in conjunction with the transport service and Ethernet/IEEE 802.3 hardware, provides complete seven layer functionality and serves as the fundamental building block for the development of a host of other services such as mail or virtual terminal (see Figure 1).

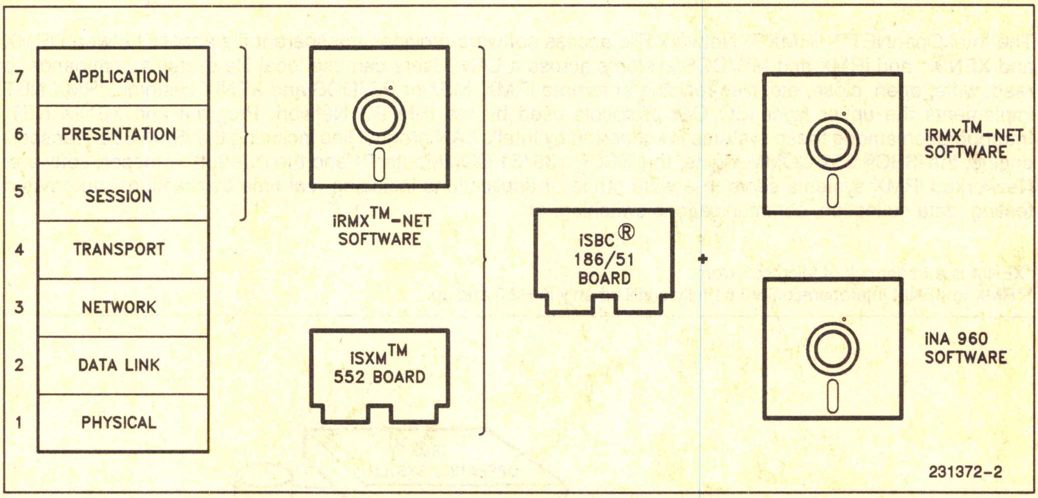


Figure 1. ISO OSI Reference Model RMX-NET

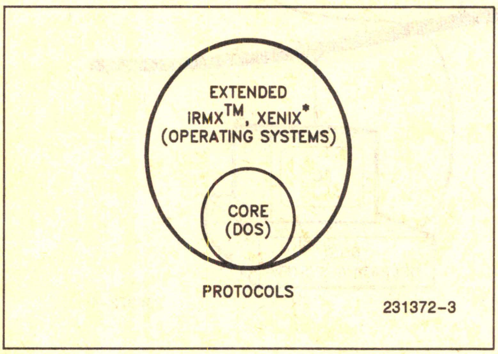


Figure 2. Protocols and Interoperation

CONSUMER	SERVER	WHICH PROTOCOL USED	SUPPORTED IN
iRMX™	iRMX™	EXT.	R1.0
iRMX™	XENIX*	EXT.	R2.0
XENIX*	iRMX™	EXT.	R2.0
XENIX*	XENIX*	EXT.	R1.0
XENIX*	DOS	CORE	R1.0
DOS	iRMX™	CORE	R1.0
DOS	XENIX*	CORE	R1.0
DOS	DOS	CORE	M/S NETWORK, IBM PC NETWORK SOFTWARE

Table 1. Protocols

TRANSPARENT REMOTE FILE ACCESS

iRMX-NET provides transparent remote file access at the BIOS, EIOS and Human Interface level. This means that all iRMX 86 applications written using BIOS, EIOS or HI commands can be used in a networked environment where the referenced files may reside at other nodes of the network.

With Release 1 of RMX-NET the user (file consumer) can transparently access files resident at remote systems configured with iRMX-NET or XENIX-NET (file servers). On the other hand, an RMX file server supports remote nodes configured with iRMX-NET, XENIX-NET, Microsoft Networks and IBM PC Network Software file consumers. For a table showing the combinations supported with the initial OpenNET product line please refer to Figure 2.

Transparent remote file access enables the user to manipulate and use remote files as if they were local. This capability can be used to develop key network services, such as mail, print server or virtual terminal with minimum additional effort.

PROTOCOLS

File sharing among different operating systems across the network is made possible through implementing a common set of file access (or file sharing) protocols under these operating systems. Network file sharing protocols are a set of rules governing the interaction between a file consumer and a file server on the same local area network. The file access protocols used by the OpenNet product line were jointly developed Intel, Microsoft and IBM.

Since the file systems of DOS, XENIX 286 and iRMX 86 are not identical, two protocol sets have been devised to support transparency in the various server-consumer combinations. The so-called "core protocols" support transparent file access between two DOS nodes on the network. The "extended protocols" support transparent file access between iRMX and XENIX nodes. The extended protocols contain the core protocols as a subset. See Figure 2 for an illustration. The core and extended protocols are in public domain and can be implemented under other operating systems, thus enabling a host of otherwise incompatible systems to share data and resources and to communicate across the network.

NETWORK HIERARCHICAL FILE SYSTEM

The file sharing protocols implemented in a network extend the file systems of the individual nodes into a so-called network hierarchical file system. Within a network any user can access each of the "public" files through a unique path of the network directory. For an illustration of the latter, please refer to Figure 3. Note that a directory can be designated as public (accessible from other nodes of the network) or private (accessible only locally) when SYSGEN-ing the server. Within a network hierarchical file system the same access right options are available as under RMX 86, that is a remote file can be read only, written into or searched depending on how it is set up.

IMPLEMENTATION

iRMX-NET implements file access across the network through introducing a new file type, the "remote file." The iRMX operating system originally supports physical, stream and named files through the respective file drivers contained within the Basic I/O system (BIOS). iRMX-NET adds a new file driver called remote file driver (RFD). All local commands referencing remote files are intercepted at the BIOS level and are redirected through the RFD to the network.

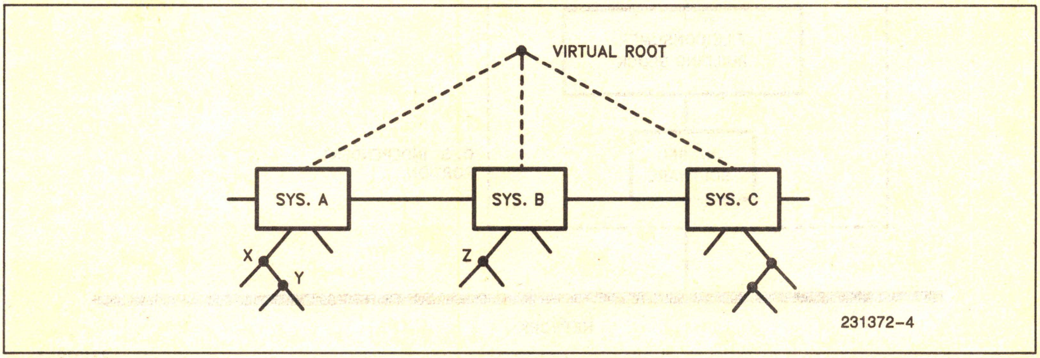


Figure 3. Network Hierarchical File System

The server receives the command from the network and forwards it to the local operating system acting as a user for the local file system. For an implementation block diagram please refer to Figures 4 and 5.

The consumer consists of two basic building blocks. The RFD is operating system dependent and must be configured to run under the host. The file consumer building block is supported by the special executive of iNA 960 and can run on a separate processor along with iNA 960.

The server includes a file server building block and a name server module which are configured to run with iNA 960 and are operating system independent. The server interfaces to the host operating system through the File Access interface which runs under the host operating system.

NAME SERVER

The Name Server provides name to network address mapping for the users. iRMX-NET implements a distributed or "protocol based" name server scheme in which every node "knows" its own name and address and thus there is no "master directory" file within the system.

When a user is referencing a remote node on the network by its name the file consumer broadcasts a request for that name across the network. The only node having the name called will respond by sending its address to the requestor.

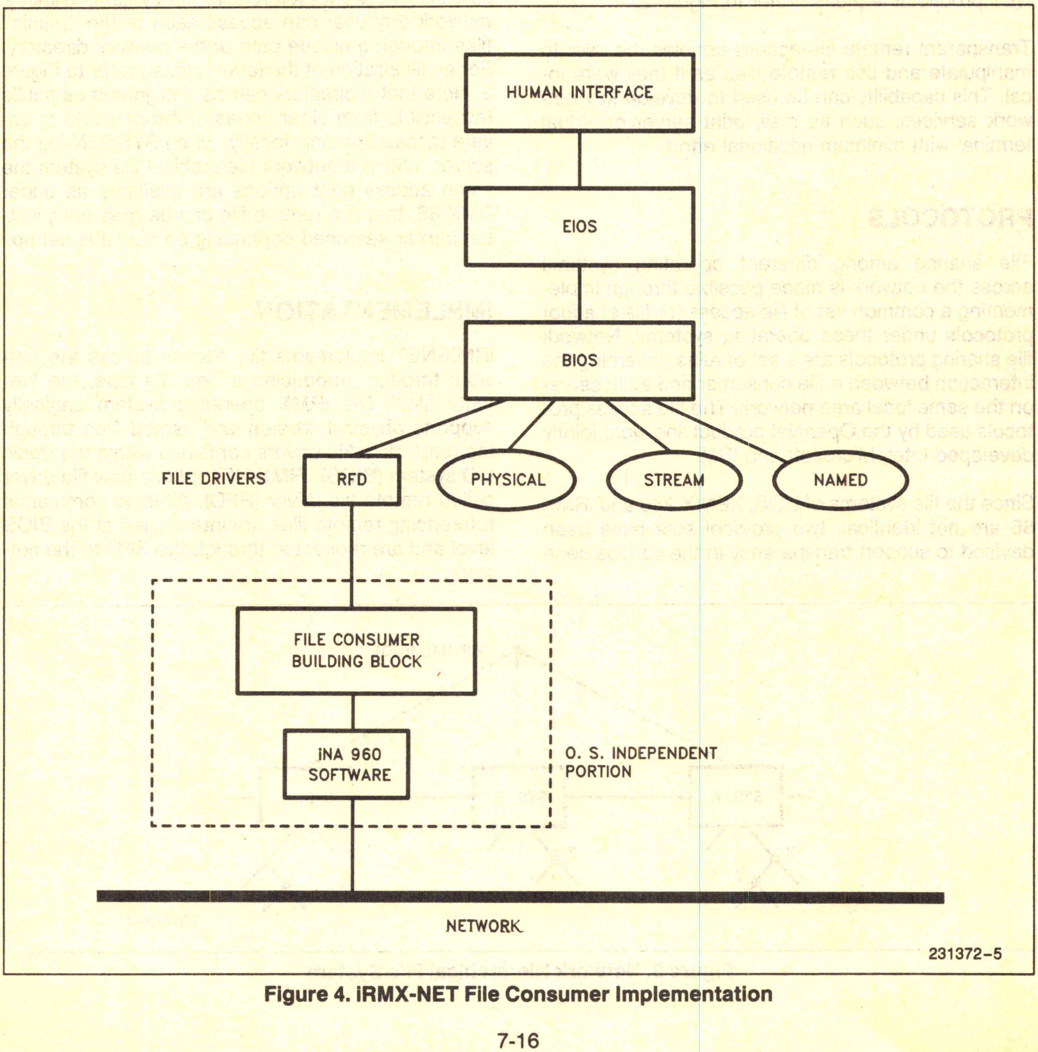


Figure 4. iRMX-NET File Consumer Implementation

SYSTEM ENVIRONMENT

iRMX-NET is supported by any system in which iRMX 86 is at release level 6.0 or later and in which the iNA 960 transport software is already configured in.

iRMX-NET is included at sysgen time as a first level job if the extended I/O system is not present or as an I/O job if it is present. iRMX-NET contains a number of user-defined parameters which must be set up when configuring the system, These parameters include the size of buffers, the number of consumers served concurrently or the maximum permissible number of outstanding processes.

USING iRMX-NET

When first referencing a remote directory the user has to issue an "attachdevice" command just like in the case of attaching a new local device under RMX 86. For example if the remote system is SYSB the user will need to issue the following command:

Attachdevice SYSB as :f5: Remote.

In this case :f5: is chosen to designate the newly opened "network volume." The "attachfile" command in fact opens a virtual circuit between the consumer and the server to support the subsequent communication between these two nodes. Once the remote device is "attached" the user can access his remote and local files alike. As a file server to a DOS consumer, iRMX-NET functions just like a PC AT file server. As a server to XENIX consumer there are a few limitations to transparency, for example, the "LOCK" and "LINK" XENIX commands are not supported under iRMX. As a file consumer to a XENIX server iRMX-NET provides full transparency.

SPECIFICATIONS

— Code size: about 40 KB

— System requirements: - RMX 86 R6.0 or later
- iNA 960

ORDERING INFORMATION

1. iRMX-NET WRO

   Object code on double density RMX diskettes with OEM license.

2. iRMX-NET WSU

   Object code on double density RMX diskettes with single user development license.

3. iRMX-NET LST

   Source listing on microfiche. (Available for R2.0 and up.)

4. iRMX-NET SRC

   Machine readable source
   (Available for R2.0 and up.)

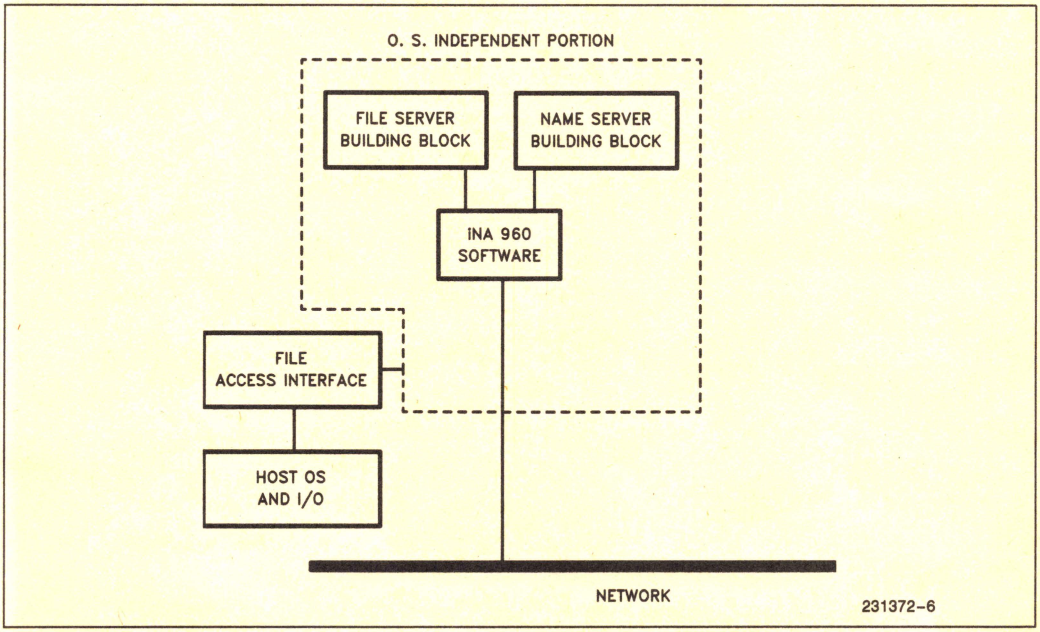


Figure 5. File Server Implementation

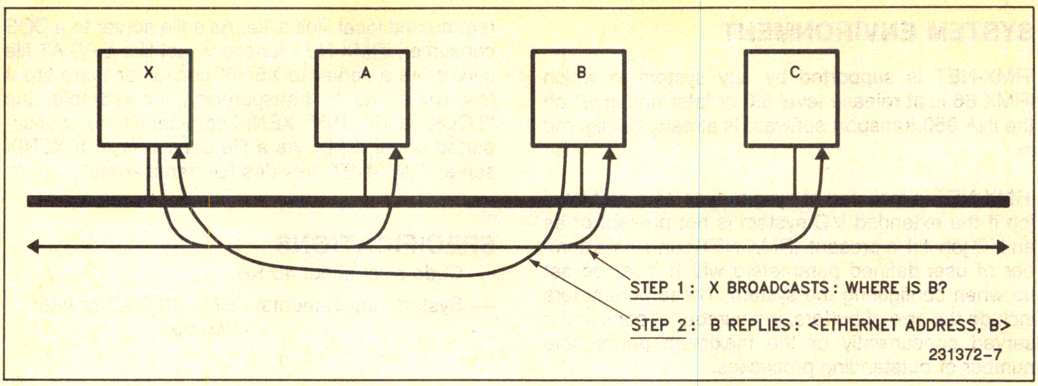


Figure 6. Distributed Name Server Scheme

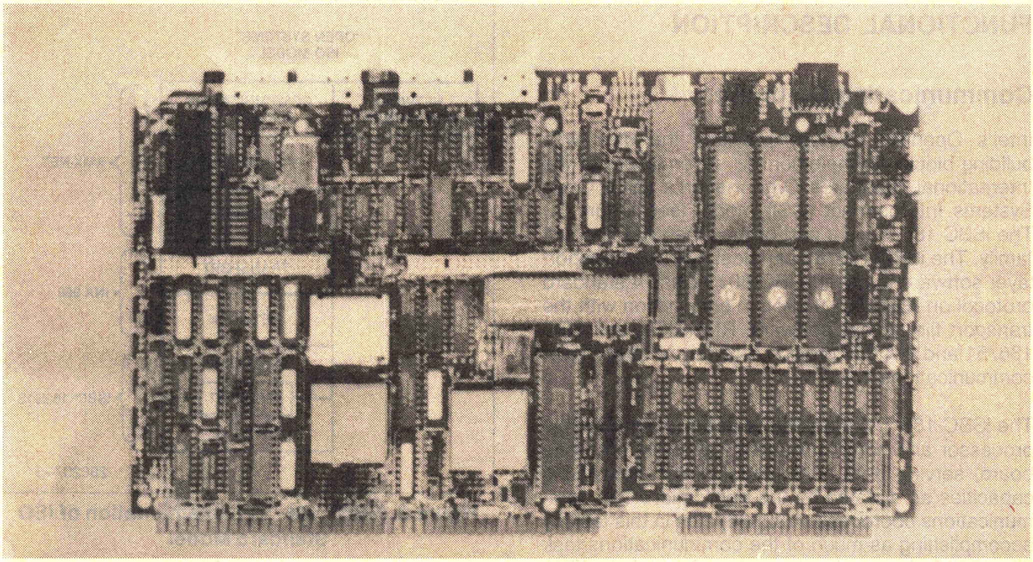
intel®

iSBC® 186/51 COMMUNICATING COMPUTER

MEMBER OF THE OpenNET™ PRODUCT FAMILY

- 6 MHz 80186 Microprocessor
- 128K Bytes of Dual-Ported RAM Expandable On-Board to 256K Bytes
- 82586 Local Area Network Coprocessor for CSMA/CD Applications and 82501 Ethernet Serial Interface for Ethernet/ IEEE 802.3 Specifications
- Two Serial Interfaces, RS-232C and RS-422A/RS-449 Compatible
- Sockets for up to 192K Bytes of JEDEC 28 Pin Standard Memory Devices
- Supports Transport Layer Software (iNA 960) and Higher Layer Communications Software (such as iRMX-NET)
- Two iSBX™ Bus Connectors
- 16M Bytes Address Range of MULTIBUS®
- MULTIBUS® Interface for Multimaster Configurations and System Expansion
- Supported by a Complete Family of Single Board Computers, Peripheral Controllers, Digital and Analog I/O, Memory, Packaging and Software

The iSBC 186/51 COMMUNICATING COMPUTER, THE COMMputer™, is a member of Intel's OpenNET family of products, and supports Intel's network software. The COMMputer utilizes Intel's VLSI technology to provide an economical self-contained computer for applications in processing and local area network control. The combination of the 80186 Central Processing Unit and the 82586 Local Area Network Coprocessor/ 82501 Ethernet Serial Interface makes it ideal for applications which require both communication and processing capabilities such as networked workstations, factory automation, office automation, communications servers, and many others. The CPU, Ethernet interface, serial communications interface, 128K Bytes of RAM, up to 192K Bytes of ROM, I/O ports and drivers and the MULTIBUS interface all reside on a single 6.75" x 12.00" printed circuit board.



280207-1

December 1986
Order Number: 280207-001

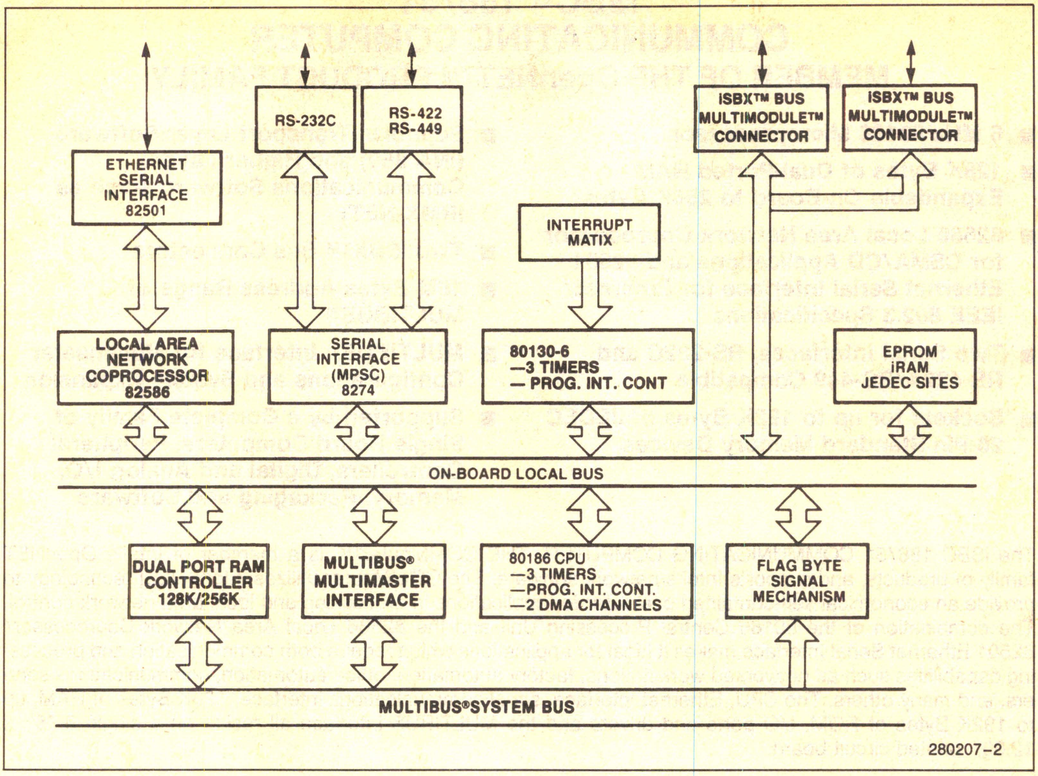


Figure 1. iSBC® 186/51 Block Diagram

FUNCTIONAL DESCRIPTION

Communicating Computer

Intel's OpenNet strategy provides the user with building blocks to implement all seven layers of the International Standards Organization's (ISO) Open Systems Interconnect (OSI) model (see Figure 2.) The iSBC 186/51 is a part of the OpenNET product family. The iSBC 186/51 can host iNA 960 transport layer software to provide ISO 8073 class 4 standard protocol on IEEE 802.3 LAN. In conjunction with the transport file access software, RMX-NET, the iSBC 186/51 and iNA 960 provide a complete seven layer communications solution.

The iSBC 186/51 board integrates a programmable processor and communications capability onto one board, serving both computational and networking capacities as dictated by the application. The communications coprocessor (82586) aids in this task by accomplishing as much of the communications task as possible before the processor intervenes (thus reducing the overhead load of the 80186 processor).

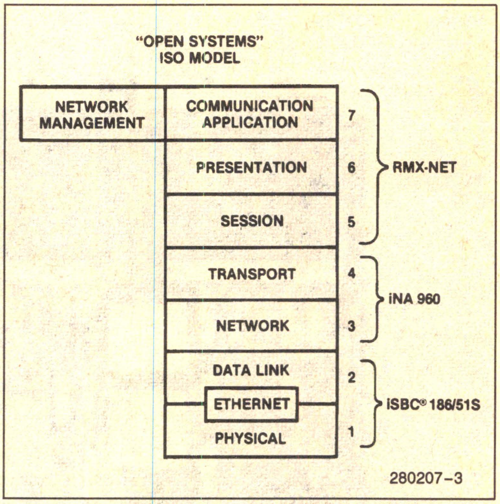


Figure 2. iSBC® 186/51 Implementation of ISO Standard Model

The dual capabilities of the iSBC 186/51 board are useful in three types of applications: (1) as a single board communicating computer running both user applications and communications tasks; (2) as one bus master of a multiple processor board solution running a portion of the overall user application and the communications tasks; and (3) as an "intelligent bus slave" that performs communications related tasks as a peripheral processor to one or more bus masters in a communications intensive environment.

Architecture

The iSBC 186/51 board is functionally partitioned into three major sections: central computer, I/O including LAN interconnect and memory including shared dual port RAM (Figure 1).

The central computer, an 80186 CPU, provides powerful processing capability. The microprocessor, together with the on-board PROM/EPROM sites, programmable timers/counters, and programmable interrupt control provide the intelligence to manage sophisticated communications operations on-board the iSBC 186/51. The timers/counters and interrupt control are also common to the I/O area providing programmable baud rates to USARTs and prioritizing interrupts generated from the USARTs. The central computer functions are protected for access by the on-board 80186 only.

The I/O is centered around the Ethernet access provided by the 82586/82501 pair. All 10 MbpS CSMA/CD protocols can be supported. Included here as well are two serial interfaces, both of which are fully programmable. In support of the single board computer, two iSBX connectors are provided for further customer expansion of I/O capabilities. The I/O is under full control of the on-board CPU and is protected from access by other system bus masters.

The third major segment, dual-port RAM memory, is the key link between the 80186, the Ethernet controller, and bus masters (if any) managing the system functions. The dual-port concept allows a common block of dynamic memory to be accessed by the on-board 80186 CPU, the on-board Ethernet controller and off-board bus masters. The system program can, therefore, utilize the shared dual-port RAM to pass command and status information between the bus masters and on-board CPU and Ethernet controllers. In addition, the dual-port concept permits blocks of data transmitted or received to accummulate in the on-board shared RAM, minimizing the need for a dedicated memory board.

CENTRAL COMPUTER FUNCTIONALITY

Central Processing Unit

The central processor for the iSBC 186/51 is Intel's 80186 CPU. The 80186 is a high integration 16-bit microprocessor. It combines several of the most common system components onto the chip (i.e., Direct Memory Access, Interval Timers, Clock generator, and Programmable Interrupt Controller). The CPU architecture includes four 16-bit Byte addressable data registers, two 16-bit index registers and two 16-bit memory base pointer registers. These are accessible by a total of 24 operand addressing modes for (1) comprehensive memory addressing, and (2) support of the data structures required for today's structured, high level languages—as well as assembly language.

Instruction Set

The 80186 instruction set is a superset of the 8086. It maintains object code compatibility while adding 10 new instructions to the existing 8086 instruction set. The 80186 retains the variable length instruction format (including double operand instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulations. Added instructions include: Block I/O, Enter and Leave subroutines, Push Immediate, Multiply Quick, Array Bounds Checking, Shift and Rotate by Immediate, and Pop and Push All.

Architectural Features

A six-byte instruction queue provides prefetching of sequential instructions and can reduce the 1000 ns minimum instruction cycle to 333 ns for queued instructions. The stack oriented architecture readily supports modular programming by facilitating fast, simple intermodule communication, and other programming constructs needed for asynchronous realtime systems. Using a windowing technique and external logic, the full 16M Bytes addressing range of the IEEE-796 MULTIBUS Standard is available to the user. The dynamic relocation scheme allows ease in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K

Bytes at a time and activation of a specific register is controlled both explicitly by program control, and implicitly by specific functions and instructions. A flag byte signaling mechanism aids in creating an interprocessor communication scheme. This includes (1) the ability to set/reset interrupts with MULTIBUS commands and (2) board reset.

Programmable Timers

The 80186 provides three internal 16-bit programmable timers. Two of these are highly flexible and are connected to four external pins (two per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source. The factory default configuration for timer 0 is baud rate generator.

The 80130-6 provides three more programmable timers. One is a factory default baud rate generator and outputs an 8254 compatible square wave to the RS232 Channel B. The other two timers are assigned to the use of the Operating System and should not be altered by the user.

The system software configures each timer independently to select the desired function. Examples of available functions are shown in Table 1. The contents of each counter may be read at any time during system operation.

Interrupt Capability

The iSBC 186/51 has two programmable interrupt controllers (PICs): one in the 80186 component and one in the 80130-6 component. In the iRMX mode, the 80186 interrupt controller acts as a slave to the 80130-6. The 80186 interrupt controller in this mode uses all of its external interrupt pins. It therefore services only internally generated interrupts (i.e., three timers, two DMA channels). The 80130-6 interrupt controller operates in the master mode and has eight prioritized inputs that can be programmed either edge or level sensitive.

The iSBC 186/51 board provides 9 vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 80186 CPU. This interrupt is typically used for signaling catastrophic events (e.g., power failure). The Programmable Interrupt Controllers (PIC) provide control and vectoring for the next eight interrupt levels. As shown in Table 2, a selection of four priority proc-

Table 1. 80186 Programmable Timer Functions

Function	Operation
Interrupt on Terminal Count	When terminal count is reached, an interrupt request is generated. This function is extremely useful for generation of real-time clocks.
Programmable One-Shot	Output goes low upon receipt of an external trigger edge or software command and returns high when terminal count is reached. This function is retriggerable.
Rate Generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-Wave Rate Generator	Output will remain high until ½ the count has been completed, and go low for the other half of the count.
Software Triggered Strobe	Output remains high until software loads count (N). N periods after count is loaded, output goes low for one input clock period.
Hardware Triggered Strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event Counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counter "window" has been enabled or an interrupt may be generated after N events occur in the system.

essing modes is available for use in designing request processing configurations to match system requirements for efficient interrupt servicing with minimal latencies. Operating modes and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from all on-board I/O resources and from the MULTIBUS system bus. The PIC then resolves requests according to the selected mode and, if appropriate, issues an interrupt to the CPU.

Interrupt Request Generation

iSBC 186/51 Interrupt Service requests may originate from 25 sources. Table 3 contains a list of devices and functions supported by interrupts. All interrupts are jumper configurable with either suitcase or wire wrap to the desired interrupt request level.

I/O FUNCTIONALITY

Local Area Network Coprocessor

The 82586 is a local communications controller designed to relieve the 80186 of many of the tasks associated with controlling a local network. The 82586 provides most of the functions normally associated with the data link and physical link layers of a local network architecture. In particular, it performs framing (frame boundary delineation, addressing, and bit error detection), link management, and data modulation. It also supports a network management interface.

The 80186 and the 82586 communicate entirely through a shared memory space. To the user, the 82586 appears as two independent but communicat-

Table 2. iSBC® 186/51 Programmable Interrupt Modes

Mode	Operation
Fully Nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest
Special Fully Nested	Allows multiple interrupts from slave PICs to the master PIC. Used in the case of cascading where the priority has to be conserved within each slave
Specific Priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment
Polled	System software examines priority-encoded system interrupt status via interrupt status register

Table 3. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS Interface	Requests from MULTIBUS resident peripherals or other CPU	2
8274	Transmit buffer empty, receive buffer full and channel errors	8
Internal 80186 PIC	Timer 0, 1, 2 outputs (function determined by timer mode) and 2 DMA channel interrupts	5
82586	Communications processor needs attention	1
Flag Byte Interrupt	Flag byte interrupt set by MULTIBUS master	1
Systick	80130-6, iRMX system timer	1
Edge to Level Trigger	Converts EDGE interrupts to level interrupts	1
iSBX Connectors MULTIMODULE	Function determined by iSBX	4 (2 per iSBX connector)
Bus Fail Safe Timer	Indicates addressed MULTIBUS resident device has not responded to command within 6 ms	1
OR-Gate Matrix	Outputs of OR-gates on-board for multiple interrupts	1

ing units: the Command Unit (CU) and the Receive Unit (RU). The CU executes the commands given by the 80186 to the 82586. The RU handles all activities related to packet reception, address recognition, CRC checking, etc. The two are controlled and monitored by the CPU via a shared memory structure called the System Control Block (SCB). Commands for the CU and RU are placed into the SCB by the host processor. Status information is placed into the SCB by the CU and RU (via the CU). The Channel Attention and Interrupt lines are used by the CPU and the 82586 to get the other to look into the SCB. See Figure 3. The 82586 features a high level diagnostic or maintenance capability. It automatically gathers statistics on CRC errors, frame alignment errors, overrun errors, and frames lost due to lack of reception resources. In addition, the user can output the status of all internal registers to facilitate system design.

Upon initialization, the 82586 obtains the address of its System Control Block through the Initialization Root which begins at location 0FFFFF6H. See Figure 4. The SCB contains control commands, status register, pointers to the Command Block List (CBL) and Receive Frame Area (RFA), and tallies for CRC, Alignment, DMA Overrun and No Resource errors. Through the SCB, the 82586 is able to provide status and error counts for the 8086, execute "programs" contained in the CBL and receive incoming frames in the Receive Frame Area (RFA).

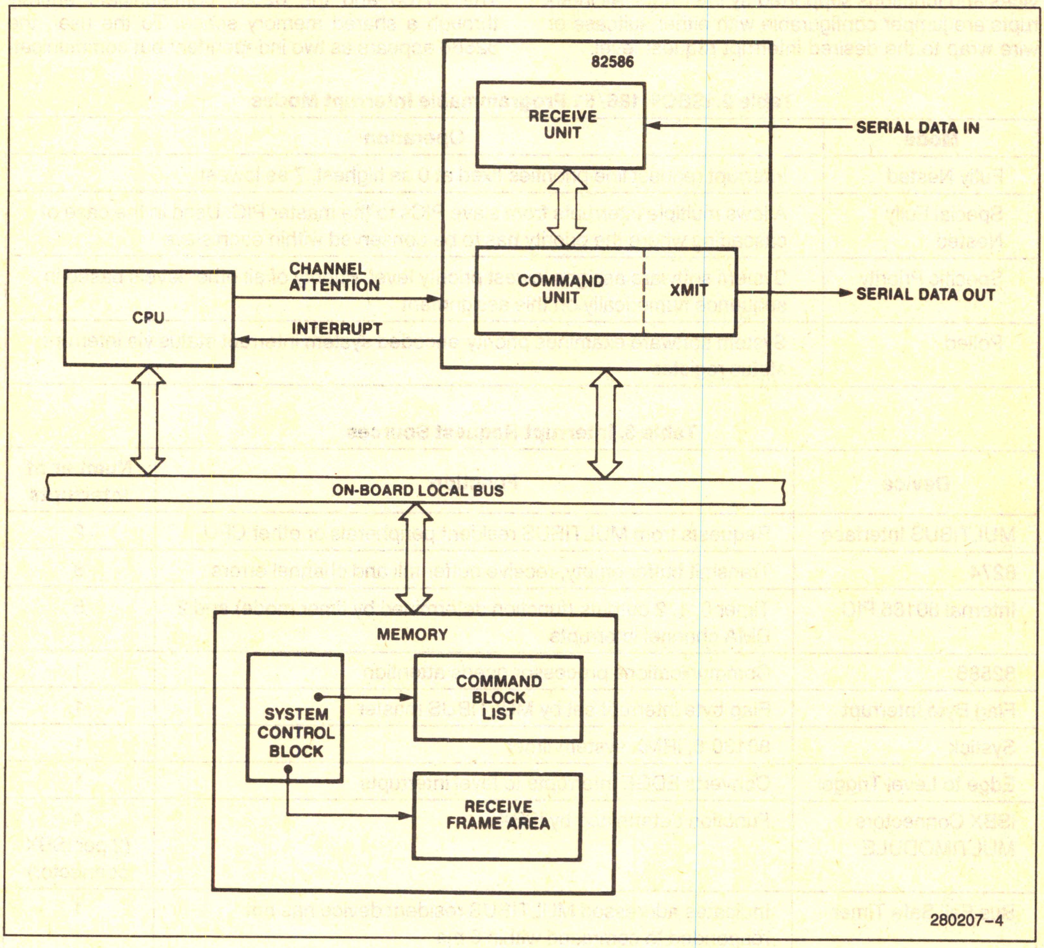


Figure 3. System Overview

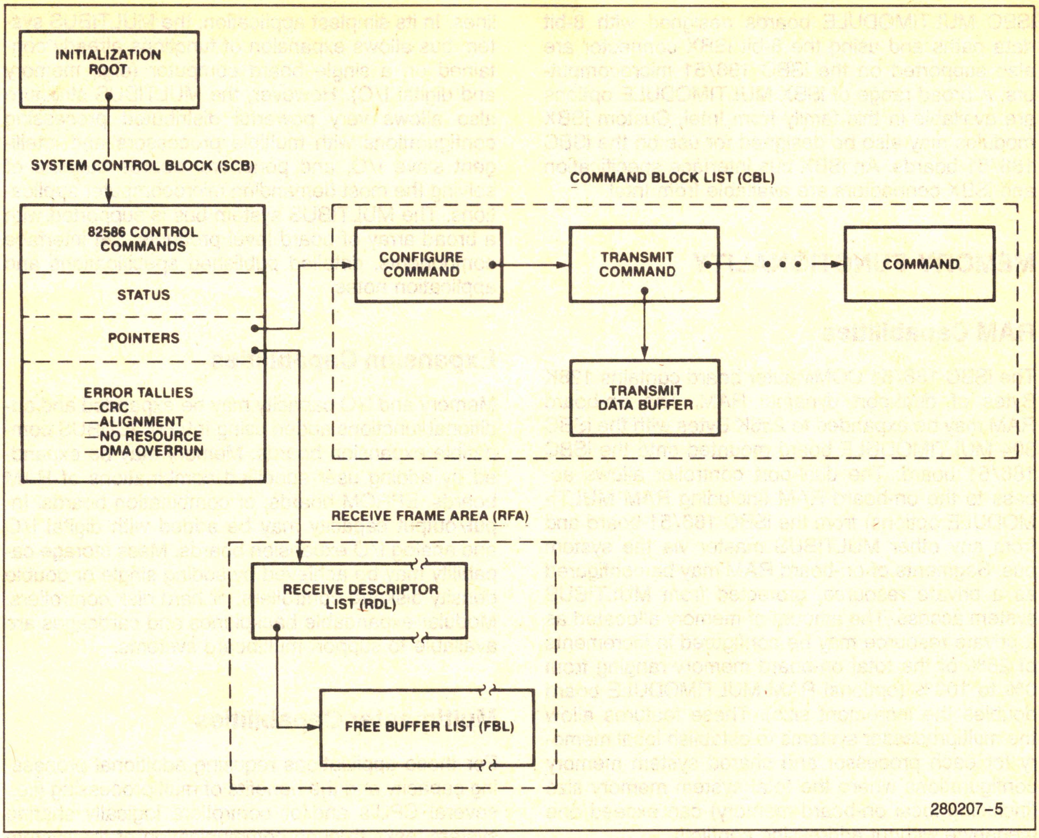


Figure 4. 82586 Memory Structures

Serial I/O

Two programmable communications interfaces using the Intel 8274 Multi-Protocol Serial Controller (MPSC) are contained on the iSBC 186/51. Two independent software selectable BAUD rate generators provide the channels with all the common communications frequencies. The mode of operation (for example, Asynchronous, Byte Synchronous or Bi-synchronous protocols), data format, control character format, parity, and baud rate are all under program control. The 8274 provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the MSPC. The iSBC 186/51 supports operation in the polled, interrupt and DMA driven interfaces through jumper options. The board is delivered previously configured with channel A in RS-422/RS-449. Channel B in RS-232C. Channel A may be configured to support RS-232C.

iSBX™ MULTIMODULE™ On-Board Expansion

Two 8/16-bit iSBX MULTIMODULE connectors are provided in the iSBC 186/51 microcomputer. Through these connectors, additional on-board I/O functions may be added. iSBX MULTIMODULE boards optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks), and other custom interfaces to meet specific needs. By mounting directly on the single board computer, less interface logic, less power, simpler packaging, higher performance, and lower cost results when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connectors on the iSBC 186/51 boards provide all signals necessary to interface to the local on-board bus, including 16 data lines for maximum data transfer rates.

iSBC MULTIMODULE boards designed with 8-bit data paths and using the 8-bit iSBX connector are also supported on the iSBC 186/51 microcomputers. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 186/51 boards. An iSBX bus interface specification and iSBX connectors are available from Intel.

MEMORY FUNCTIONALITY

RAM Capabilities

The iSBC 186/51 COMMputer board contains 128K Bytes of dual-port dynamic RAM. The on-board RAM may be expanded to 256K Bytes with the iSBC 304 MULTIMODULE board mounted onto the iSBC 186/51 board. The dual-port controller allows access to the on-board RAM (including RAM MULTIMODULE options) from the iSBC 186/51 board and from any other MULTIBUS master via the system bus. Segments of on-board RAM may be configured as a private resource, protected from MULTIBUS system access. The amount of memory allocated as a private resource may be configured in increments of 25% of the total on-board memory ranging from 0% to 100% (optional RAM MULTIMODULE board doubles the increment size). These features allow the multiprocessor systems to establish local memory for each processor and shared system memory configurations where the total system memory size (including local on-board memory) can exceed one megabyte without addressing conflicts.

Universal Memory Sites for Local Memory

Six 28-pin sockets are provided for the use of Intel's 2732, 2764, 27128, 27256 EPROMs and their respective ROMs. When using the 27256s, the on-board EPROM capacity is 192K Bytes. Other JEDEC standard pinout devices are also supported, including byte-wide static RAMs and iRAMs.

MULTIBUS® SYSTEM BUS AND MULTIMASTER CAPABILITIES

Overview

The MULTIBUS system bus is Intel's industry standard microcomputer bus structure. Both 8- and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In its simplest application, the MULTIBUS system bus allows expansion of functions already contained on a single board computer (e.g., memory and digital I/O). However, the MULTIBUS structure also allows very powerful distributed processing configurations with multiple processors and intelligent slave I/O, and peripheral boards capable of solving the most demanding microcomputer applications. The MULTIBUS system bus is supported with a broad array of board level products, LSI interface components, detailed published specifications and application notes.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be added with digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and cardcages are available to support multiboard systems.

Multimaster Capabilities

For those applications requiring additional processing capacity and the benefits of multiprocessing (i.e., several CPU's and/or controllers logically sharing system tasks through communication of the system bus), the iSBC 186/51 boards provide full MULTIBUS arbitration control logic. This control logic allows up to three iSBC 186/51 boards or other bus master, including iSBC 80 family MULTIBUS compatible 8-bit single board computers to share the system bus using a serial (daisy chain) priority scheme. This allows up to 16 masters to share the MULTIBUS system bus with an external parallel priority decoder. In addition to the multiprocessing configurations made possible with multimaster capability, it also provides a very efficient mechanism for all forms of DMA (Direct Memory Access) transfers.

MISCELLANEOUS FUNCTIONALITY

Power-Fail Control and Auxiliary Power

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the

protection of RAM contents during system power-down sequences. An auxiliary power bus is also provided to allow separate power to RAM for systems requiring battery back-up of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

System Development Capabilities

The development cycle of iSBC 186/51 products can be significantly reduced and simplified by using either the System 86/3XX or the Intellec Series Microcomputer Development Systems. The Assembler, Locating Linker, Library Manager, Text Editor and System Monitor are all supported by the ISIS-II disk-based operating system. To facilitate conversion of the 8080A/8085A assembly language programs to run on the iSBC 186/51 boards, CONV-86 is available under the ISIS-II operating system.

In-Circuit Emulator

The Integrated Instrumentation In-Circuit Emulator (I^2ICE) provides the necessary link between the software development environment provided by the Intellec system and the "target" iSBC 186/51 execution system. In addition to providing the mechanism for loading excutable code and data into the iSBC 186/51 boards, the I^2ICE-186 provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software.

PL/M-86 and C-86

Intel has two systems implementation languages, PL/M-86 and C-86. Both are standard in the System 86/3XX and are also available as Intellec Microcomputer Development System options. PL/M-86 provides the capability to program in algorithmic language and eliminates the need to manage register usage or allocate memory while still allowing explicit control of the system's resources when needed. C-86 is especially appropriate in applications requiring portability and code density. FORTRAN 86 and PASCAL 86 are also available on Intellec or 86/3XX systems.

Run-Time Support

Intel also offers two run-time support packages: iRMX 88 Realtime Multitasking Executive and the iRMX 86 Operating System. The iRMX 88 executive is a simple, highly configurable and efficient foundation for small, high performance applications. Its multitasking structure establishes a solid foundation for modular system design and provides task scheduling and management, intertask communication and synchronization, and interrupt servicing for a variety of peripheral devices. Other configurable options include terminal handlers, disk file system, debuggers and other utilities. The iRMX 86 Operating System is a highly functional operating system with a very rich set of features and options based on an object-oriented architecture. In addition to being modular and configurable, functions beyond the nucleus include a sophisticated file management and I/O system, and a powerful human interface. Both packages are easily customized and extended by the user to match unique requirements.

SPECIFICATIONS

Word Size

Instruction: 8, 16, 24, or 32 bits
Data: 8, 16 bits

System Clock

6.00 MHz ± 0.1%

Cycle Time

Basic Instruction Cycle

6 MHz— 1000 ns
333 ns (assumes instruction in the queue)

NOTE:
Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles.)

Memory Capacity/Addressing

Six Universal Memory Sites support JEDEC 24/28 pin EPROM, PROM, iRAM and static RAM.

Example for EPROM:

Device	Total Capacity	Address Range
2732	24K Bytes	$F8000-FFFFF_H$
2764	48K Bytes	$F0000-FFFFF_H$
27128	96K Bytes	$E0000-FFFFF_H$
27256	192K Bytes	$C0000-FFFFF_H$

On-Board RAM

Board	Total Capacity	Address Range
iSBC 186/51	128K Bytes	$0-1FFFF_H$

With MULTIMODULE™ RAM

Board	Total Capacity	Address Range
iSBC 304	256K Bytes	$0-3FFFF_H$

I/O Capacity

Serial—two programmable channels using one 8274
iSBX MULTIMODULE—two 8/16-bit iSBX connectors allow use of up to 2 single-wide modules or 1 single-wide module and 1 double-wide iSBX module.

Serial Communications Characteristics

Synchronous — 5–8 bit characters; internal or external character synchronization; automatic sync insertion

Asynchronous — 5–8 bit characters; break character after generation; 1, ½, or 2 stop bits; false start bit detection

Baud Rates

Frequency (KHz) (S/W Selectable)	Baud Rate (Hz)		
	Synchronous	Asynchronous	
	÷1	÷16	÷64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38,400	2400	600
19.2	19,200	1200	300
9.6	9,600	600	150
4.8	4,800	300	75
2.4	2,400	150	—
1.76	1,760	110	2400

NOTE:
Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (80186 timer 0 and 80130 baud timer).

Timers

Input Frequencies

Reference 1.5 MHz ±0.1% (0.5 μs period nominal)
Event Rate: 1.5 MHz max.

80186 Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual (Cascaded) Timer/Counter	
	Min	Max	Min	Max
Real-Time Interrupt	667 ns	43.69 ms	667 ns	47.72 minutes
Programmable One-Shot	1000 ns	43.69 ms	1000 ns	47.72 minutes
Rate Generator	22.889 Hz	1.5 MHz	0.0003492 Hz	1.5 MHz
Square-Wave Rate Generator	22.889 Hz	1.5 MHz	0.0003492 Hz	1.5 MHz
Software Triggered Strobe	1000 ns	43.69 ms	1000 ns	47.72 minutes
Event Counter	—	1.5 MHz	—	—

Interfaces

Ethernet— IEEE 802.3 compatible

MULTIBUS®— IEEE 796 compatible

MULTIBUS®— Master D16 M24 I16 V0 EL

Compliance

iSBX™ Bus— IEEE P959 compatible

Serial I/O— RS-232C compatible, configurable as a data set or data terminal, RS-422A/RS-449

Connectors

Interface	Double-Sided Pins	Centers (in.)	Mating Connectors
Ethernet	10	0.1	AMP87531-5
MULTIBUS SYSTEM	86 (P1)	0.156	Viking 3KH43/9AMK12 Wire Wrap
	60 (P2)	0.1	Viking 3KH30/9JNK
iSBX Bus 8-Bit Data	36	0.1	iSBX 960-5
16-Bit Data	44	0.1	iSBX 960-5
Serial I/O	26	0.1	3M 3452-0001 Flat or AMP88106-1 Flat

Physical Characteristics

Width: 12.00 in. (30.48 cm)

Height: 6.75 in. (17.15 cm)

Depth: 0.70 in. (1.78 cm)

Weight: 18.7 ounces (531 g.)

Environmental Characteristics

Operating Temperature: 0°C to 55°C

Relative Humidity: 10% to 90% (without condensation)

Electrical Characteristics

DC Power Supply Requirements

Configuration	Maximum Current (All Voltages ±5%)		
	+5	+12	−12
SBC 186/51 as shipped:			
Board Total	7.45A	40 mA	40 mA
With separate battery back-up	6.30A	40 mA	40 mA
Battery back-up	1.15A	—	—
With SBC-304 Memory Module Installed:			
Board Total	7.55A	40 mA	40 mA
With separate battery back-up	6.30A	40 mA	40 mA
Battery back-up	1.25A	—	—

NOTES:
1. Add 150 mA to 5V current for each device installed in the 6 available Universal Memory Sites.
2. Add 500 mA to 12V current if Ethernet transceiver is connected.
3. Add additional currents for any SBX modules installed.

Reference Manual

122330-001—iSBC 186/51 Hardware Reference Manual (NOT SUPPLIED)

Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

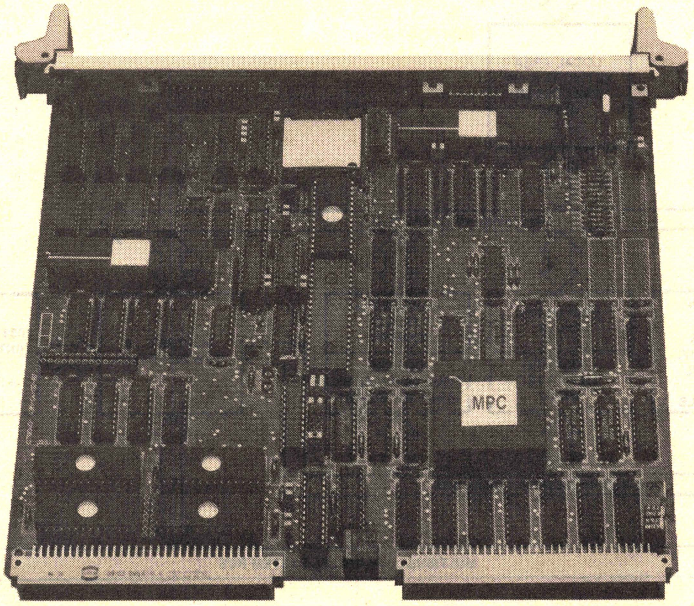
Ordering Information

Part Number	Description
SBC 186/51	Communicating Computer

iSBC® 186/530 MULTIBUS® II ETHERNET (IEEE 802.3) COMMUNICATIONS ENGINE

- Provides ETHERNET (IEEE 802.3) Compatible Networking Capability for all MULTIBUS® II Systems
- High Integration 8 MHz 80186 Microprocessor
- 256K Bytes DRAM Provided, with Sockets to Expand to 512K Bytes DRAM On-Board
- MULTIBUS II iPSB (Parallel System Bus) Interface with Full Message Passing Capability
- Host Operating System Independent
- Four 28-Pin JEDEC Sites, Expandable to 8 Sites with iSBC® 341 MULTIMODULE™ for a Maximum of 512K Bytes EPROM
- Provides one RS232C Serial Port for Use in Debug and Testing
- MULTIBUS II Interconnect Space for Software Configurability and Diagnostics
- Resident Firmware to Support Built-in-Self-Test (BIST) Power-up Diagnostics, and Host-To-Controller Software Download

The iSBC® 186/530 MULTIBUS® II ETHERNET (IEEE 802.3) Communications Engine is a dedicated ETHERNET communications front-end processor implementing the full, high performance message passing interface of the MULTIBUS II (iPSB) Parallel System Bus. This iSBC board combines an 8 MHz 80186 16-bit microprocessor, an 82586 Local Area Network Coprocessor, an Ethernet Serial Interface component, up to 512K bytes of DRAM, four 28-pin JEDEC sites, and one RS232C serial port on a single 220 mm × 233 mm (8.7 in. × 9.2 in.) Eurocard printed circuit board. Acting as a communications engine, the iSBC 186/530 board off-loads the host CPU(s) in a MULTIBUS II system from managing and executing Ethernet LAN communications tasks. The main advantage of the communications engine concept is the ability to add IEEE 802.3 networking capability to a MULTIBUS II system without requiring a major design effort. The features of the board create a flexible, intelligent communications controller capable of supporting off-the-shelf or custom configurations on IEEE 802.3 LANs.



280269-1

October 1986
Order Number: 280269-001

FUNCTIONAL DESCRIPTION

Overview

The iSBC 186/530 MULTIBUS II ETHERNET Communications Engine is a powerful IEEE 802.3 LAN communications sub-system specifically designed to operate in and support message-based, multiprocessor system configurations being implemented on the MULTIBUS II architecture. The board's on-board CPU, an 8 MHz 80186 microprocessor, provides significant intelligence to off-load and distribute the LAN communications functions away from one or all of a system's processor boards.

The iSBC 186/530 board was designed as a dedicated ETHERNET LAN front-end processor to enable the OEM to connect MULTIBUS II-based systems with different operating systems to the same network.

ARCHITECTURE

The iSBC 186/530 board supports the full iPSB bus interface functions of data and interrupt message passing, interconnect space, memory space, and I/O references. This board supports both requestor and replier functions as described in the MULTIBUS II Architecture Specification Handbook (#146077, Rev. C). The board consists of six major subsystem areas: Processor, ETHERNET I/O, Memory, General I/O, iPSB bus Interface, and Interconnect (See Figure 1).

Processor Subsystem

80186 PROCESSOR

The central processor unit on the iSBC 186/530 board is Intel's 16-bit 8 MHz 80186 microprocessor.

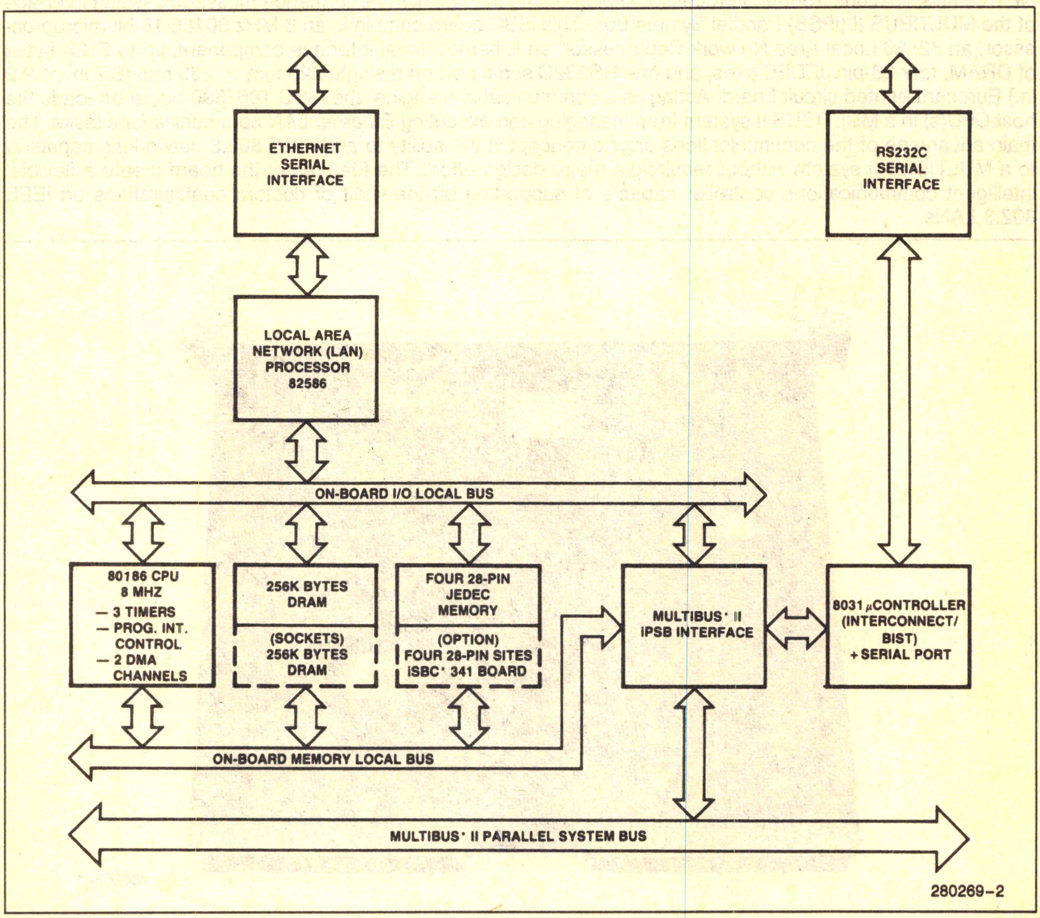


Figure 1. iSBC® 186/530 Board Functional Block Diagram

The highly integrated 80186 CPU combines several system components onto a single chip (i.e., two Direct Memory Access lines, three Interval Timers, Clock Generator, and Programmable Interrupt Controller). The 80186 instruction set is a superset of the 8086 and maintains object code compatibility while adding additional instructions. This high performance component provides the intelligent interface between engine and host processor(s) and manages the board's LAN communications capability. Refer to the Microsystem Components Handbook, Order Number 230843-00X, for more detailed information on the hardware operation and requirements of the 80186 microprocessor component.

DIRECT MEMORY ACCESS (DMA) FUNCTION

The iSBC 186/530 board uses the 80186 microprocessor to provide two DMA channels for DMA support of the iPSB bus interface, the MPC Message Passing Coprocessor chip (See Table 1).

Table 1. iSBC® 186/530 Board DMA Channel Allocation

DMA Configuration (80186)	
80186	**Local Bus Resource**
DMA Channel 0	Output DMA to MPC
DMA Channel 1	Input DMA from MPC (Message Passing Coprocessor)

ETHERNET I/O Subsystem

The ETHERNET interface on the iSBC 186/530 board is implemented by the 82586 LAN coprocessor and the Ethernet Serial Interface component. Data is transferred between the on-board memory of the iSBC 186/530 board and the 82586 controller by 82586 initiated DMA. The 82586 initiates the DMA cycles by activating the HOLD signal to the 80186 processor. The DMA cycle begins when the 80186 processor activates the HOLD ACKNOWLEDGE signal.

The 82586 component provides most of the functions normally associated with the data link and physical link layers of a local network architecture (See Figure 2). In particular, it performs framing (frame boundary delineation, addressing, and bit error detection), link management, and data modulation. It also supports a network management interface.

The Ethernet Serial Interface component performs Manchester encoding and decoding of the transmit and receive frames. It also provides the electrical interface to the Ethernet transceiver cable. Both chips support a loop-back function. The pin assignments for the Ethernet connector are shown in Table 2.

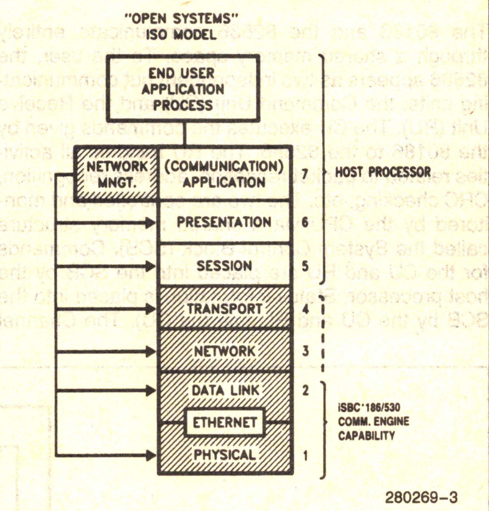


Figure 2. ISO Layered Model and the iSBC® 186/530 Board

Table 2. ETHERNET Connector, Pin Assignments

Pin	Description	Pin	Description
1	Shield	9	Collision (−)
2	Collision (+)	10	Transmit (−)
3	Transmit (+)	11	Reserved
4	Reserved	12	Receive (−)
5	Receive (+)	13	Power
6	Power Return	14	Reserved
7	Reserved	15	Reserved
8	Reserved		

Each iSBC 186/530 board is manufactured with a unique default 48-bit Ethernet network address

stored in an address PROM. This address PROM is protected by checksum and can be read by utilizing the on-board I/O space. The 82586 component can be programmed to have this or any other Ethernet address.

80186/82586 COMMUNICATION

The 80186 and the 82586 communicate entirely through a shared memory space. To the user, the 82586 appears as two independent but communicating units: the Command Unit (CU) and the Receive Unit (RU). The CU executes the commands given by the 80186 to the 82586. The RU handles all activities related to packet reception, address recognition, CRC checking, etc. The two are controlled and monitored by the CPU via a shared memory structure called the System Control Block (SCB). Commands for the CU and RU are placed into the SCB by the host processor. Status information is placed into the SCB by the CU and RU (via the CU). The Channel Attention and Interrupt lines are used by the CPU and the 82586 to get the other to look into the SCB (See Figure 3).

The 82586 features a high level diagnostic or maintenance capability. It automatically gathers statistics on CRC errors, frame alignment errors, overrun errors, and frames lost due to lack of reception resources. In addition, the user can output the status of all internal registers to assist in system design.

Upon initialization, the 82586 obtains the address of its System Control Block through the Initialization Root which begins at location 0FFFFF6H (See Figure 4). The SCB contains control commands, status register, pointers to the Command Block List (CBL) and Receive Frame Area (RFA), and tallies for CRC, Alignment, DMA Overrun, and No Resource errors. Through the SCB, the 82586 is able to provide status and error counts for the 80186, execute "programs" contained in the CBL and receive incoming frames in the Receive Frame Area (RFA).

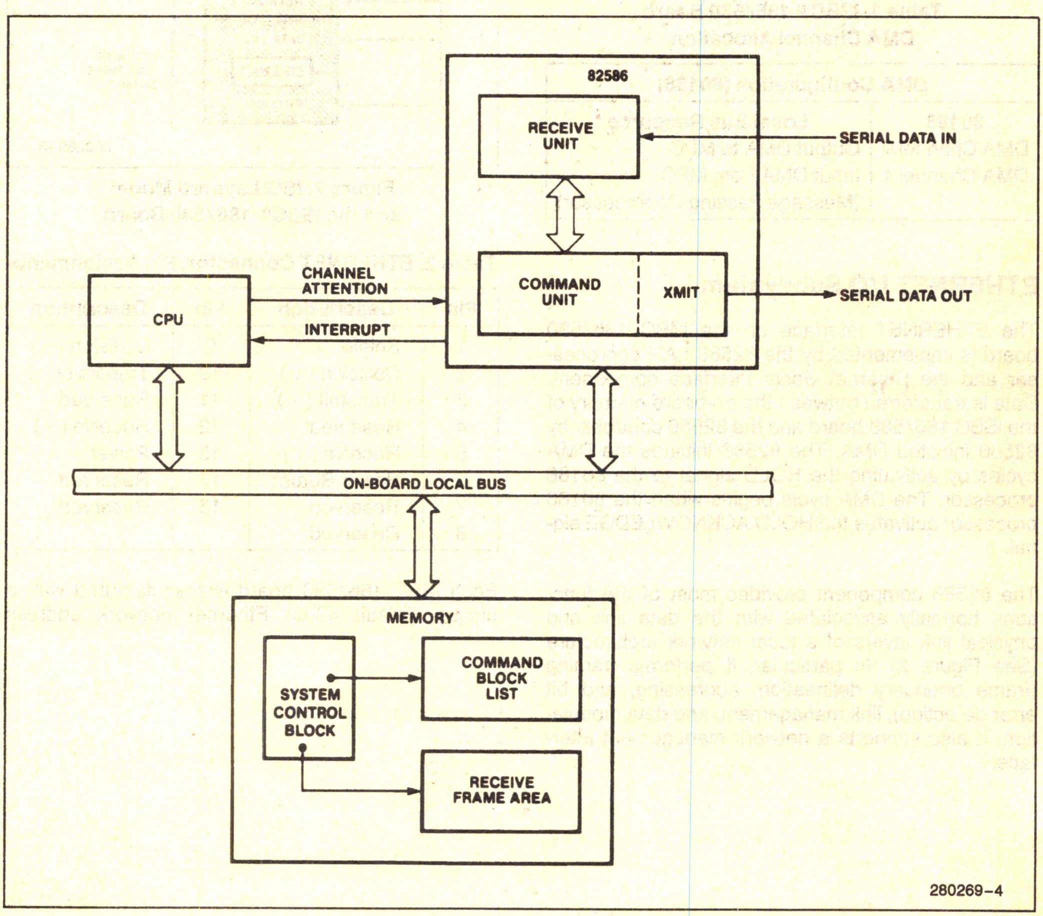


Figure 3. System Overview

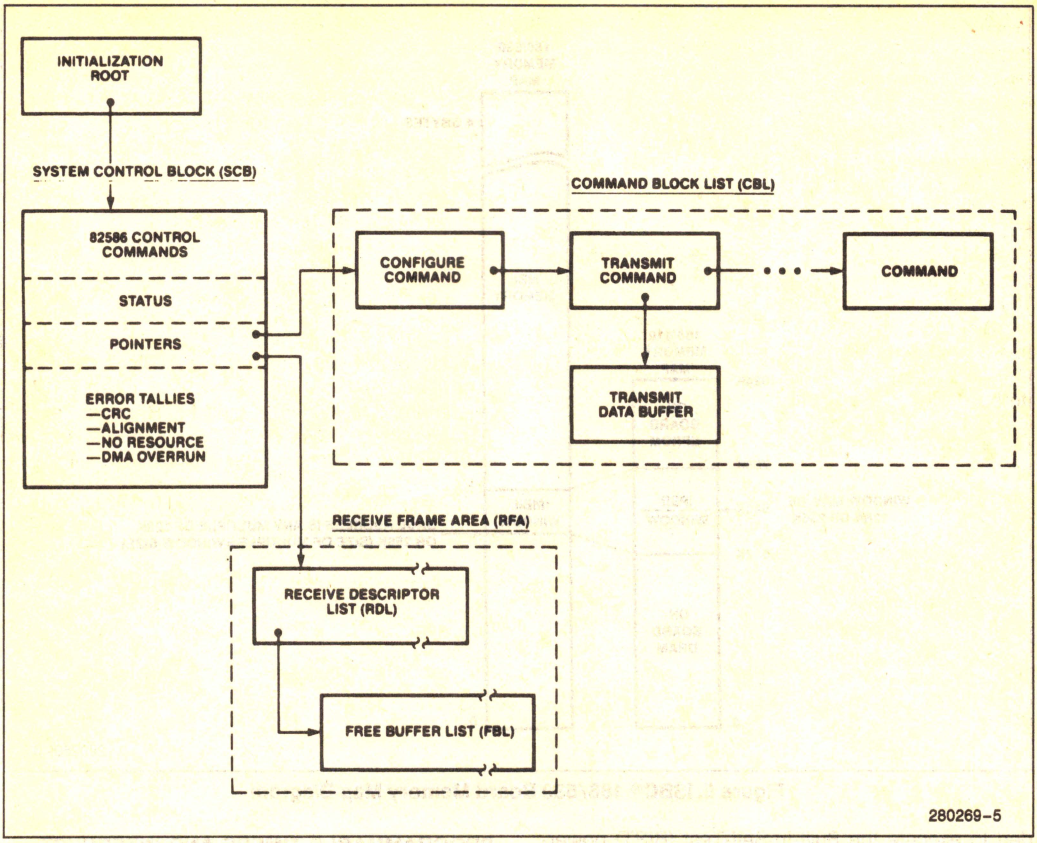


Figure 4. 82586 Memory Structures

Memory Subsystem

The iSBC 186/530 board's on-board memory subsystem consists of a large DRAM array and a set of ROM/EPROM memory sites. Access to the on-board memory subsystem resources, as well as off-board iPSB bus access, is accomplished by observing the iSBC 186/530 board memory map (See Figure 5). The mapping occurs within the 1 megabyte memory space of the 80186 microprocessor, and is split into three main areas: DRAM reserved, iPSB window, and EPROM reserved. The first 0 to 512K bytes is always reserved for local DRAM, the next 128K or 256K bytes (or up to 768K) is the iPSB window, and the remaining 384K or 256K byte area is reserved for local EPROM. The iPSB window maps a 128K or 256K byte memory area into the 4 gigabyte global physical address range of the MULTIBUS II iPSB bus. This window is programmable and allows the 80186 processor to access the complete 4 gigabyte memory space of the iPSB bus.

The board's memory map also supports a 64K byte access window for I/O space between local and iPSB bus access. The 64K bytes of local I/O space is mapped 1-to-1 to the iPSB bus' 64K byte I/O space. The upper 32K bytes access the iPSB bus I/O space, and the lower 32K bytes are reserved for local on-board I/O.

DRAM CAPABILITIES

The iSBC 186/530 board comes standard with a 256K byte DRAM memory array on-board. Eight additional 18-pin sockets are provided to the OEM for expanding the DRAM array to 512K bytes.

EPROM MEMORY

A total of four 28-pin JEDC universal sites reside on the iSBC 186/530 board. These sockets support additon of byte-wide ROM and EPROM devices in densites from 8K bytes (2764) to 64K bytes (27512) per device. Two of the four sockets contain a pair of 27128 EPROM devices installed at the factory. These devices contain 32K bytes of firmware provi-

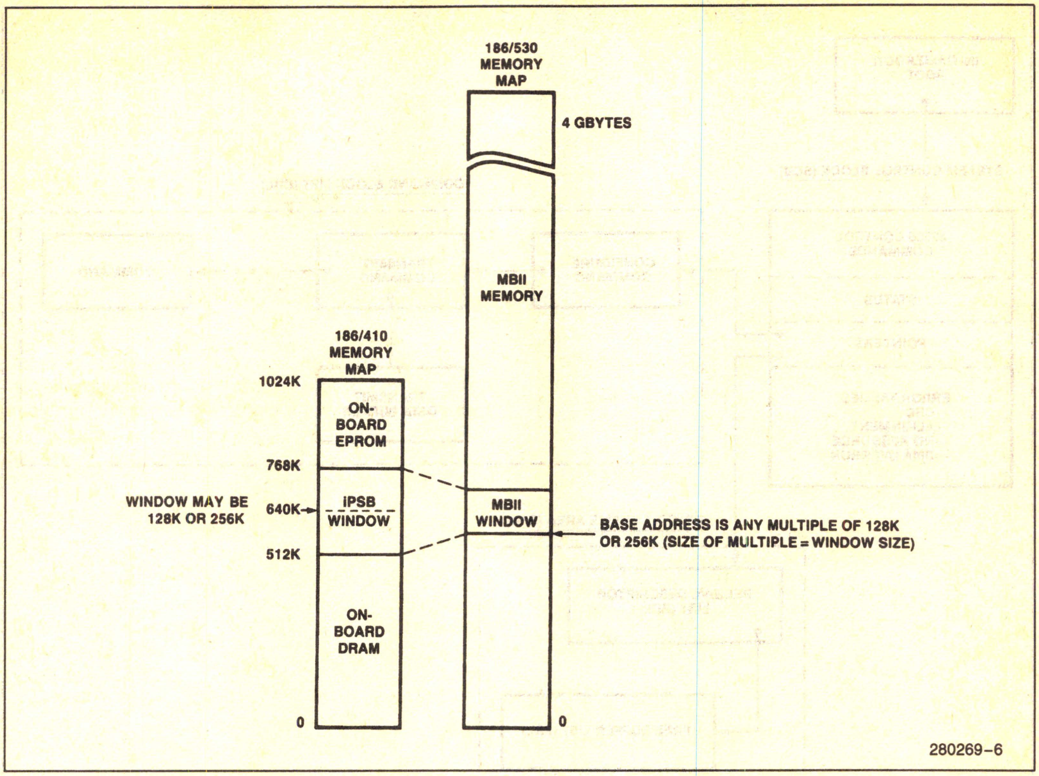


Figure 5. iSBC® 186/530 Board Memory Map Diagram

ded to execute the Built-In-Self-Test (BIST) power-up diagnostics routine, EPROM devices installed at the factory. These devices contain 32K bytes of firmware provided to execute the Built-In-Self-Test (BIST) power-up diagnostics routine. The remaining two sockets allow the user to add either two ROM/EPROM devices or an iSBC 341 256K byte EPROM MULTIMODULE™ board for a maximum of 512K bytes of ROM/EPROM on-board.

General I/O Subsystem

The I/O subsystem provides timers, interrupt control and an RS232C serial port for debug and test.

PROGRAMMABLE TIMERS AND INTERRUPT CONTROL

The board's 80186 microprocessor provides three independent, fully programmable 16-bit interval timers/event counters and an interrupt controller.

The 80186 interrupt controller is configured in the "fully nested mode," and supports five external interrupt sources via five dedicated pins provided on the 80186. All five pins are used as interrupt requests from other hardware on-board (See Table 3).

Table 3. External Interrupt Sources

Interrupt	Vector Type	Vector Location	Default Priority	Function
NMI	2	00008 H	1	Reset stake pin
INT0	12	00030 H	6	Interrupt from the Ethernet Controller
INT1	13	00034 H	7	Message Interrupt from the MPC (MINT)
INT2	14	00048 H	8	Error Interrupt from the MPC (EINT)
INT3	15	0004C H	9	Interrupt from the 8031 Interconnect Controller

RS232C SERIAL PORT

There is a simple RS232C serial port provided on the iSBC 186/530 board for use in debug and test. The serial interface is derived from the 8031 serial interface port. Only the Receive Data (RD) and Transmit Data (TD) lines are supported, connected to a 25-pin connector on the front panel. The pin assignments for the 25-pin connector are shown in Table 4.

Table 4. Serial Interface Connector, Pin Assignments

Pin	RS232CFunction	Pin	RS232C Function
1	Shield	14	Not used
2	Transmit Data (T×D)	15	Not used
3	Receive Data (R×D)	16	Not used
4	Not Used	17	Not Used
5	Not Used	18	Not Used
6	Not Used	19	Not Used
7	Signal Ground (0V)	20	Not Used
8	Not Used	21	Not Used
9	Not Used	22	Not Used
10	Not Used	23	Not Used
11	Not Used	24	Not Used
12	Not Used	25	Not Used
13	Not Used		

iPSB Bus Interface Subsystem

This subsystem's main component is the MPC Message Passing Coprocessor chip. Subsystem services provided by the MPC bus interface component includes full message, memory, I/O, and interconnect access to the iPSB bus by the 80186 and 82586 processors.

The single-chip Message Passing Coprocessor is a highly integrated CMOS device implementing the full message passing protocol and performing all the arbitration, transfer, and exception cycle protocols specified in the MULTIBUS II Architecture Specification Handbook, Rev. C., Order Number 146077.

Interconnect Subsystem

MULTIBUS II interconnect space is a standardized set of read-only and software configurable registers designed to hold and control board configuration information, and communicate system and board level diagnostics and testing information. Interconnect space is implemented with an 8031 microcontroller and the MPC silicon resident on the iSBC 186/530 board.

The read-only registers store information such as, board type, vendor I.D., firmware rev. level, etc. The software configurable registers are used for auto-software configurability and remote/local diagnostics and testing. For example, a software monitor can be used to dynamically change bus memory sizes, enable on-board resources such as memory, read if the PROM devices are installed, or access results of Built-In-Self-Tests and other diagnostics.

Most options on the iSBC 186/530 board are controlled by interconnect space. In addition, many of the interconnect registers on the board perform functions traditionally done by jumper stakes. Other interconnect registers provide status information allowing system software to determine configuration status.

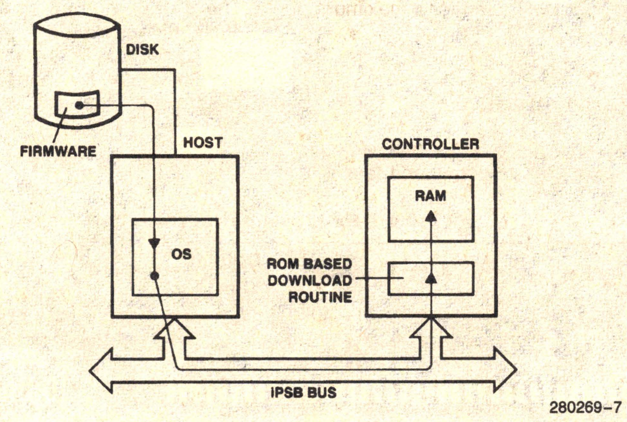


Figure 6. Download Routine

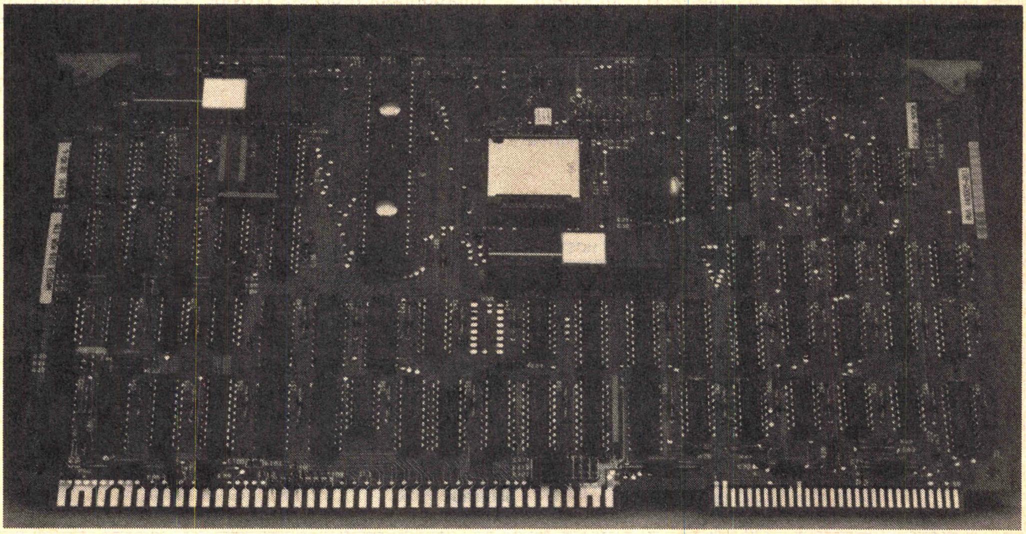
intel®

iSBC® 552A AND iSXM™ 552A IEEE 802.3 COMPATIBLE COMMUNICATIONS ENGINE PRODUCTS MEMBER OF THE OpenNET™ PRODUCT FAMILY

- **Provides High-Performance Network Front-End Processing for All MULTIBUS® I Systems Regardless of the Operating System of the Host**
  - **—Intelligent Controller with an 8 MHz 80186 Processor and 256K of DRAM Memory**
  - **—IEEE 802.3 Network Port Driven by the 82586 LAN Coprocessor**
- **Can Execute On-Board the Intel iNA 960/961 Software, an Implementation of Industry Standard ISO 8073 Transport and ISO 8473 Network Protocols**
- **Resident Network Software Can be Down-Loaded Over the Bus or the LAN**
- **On-Board Diagnostic and Boot Firmware**
- **Supported by XNX-NET and RMX-NET Network File Service Software Products**
- **Available in Two Versions**
  - **—iSBC 552A is a Flexible, Intelligent Communications Controller for IEEE 802.3 LANs**
  - **—iSXM™ 552A is a Preconfigured Controller for Executing iNA 961 Transport and Network Software as a Fully Qualified System Extension Module for the System 310 Family Products**

The iSBC 552A and iSXM 552A COMMengine products are designed for communications front end processor applications connecting MULTIBUS I systems onto IEEE 802.3 compatible LANs. COMMengines are dedicated to the communications tasks within a system allowing the host to spend more time processing user applications. A major advantage of COMMengines is that they can be used to network existing systems and established designs without forcing the redesign of the entire system architecture.

The iSBC and iSXM 552A boards can be used with any operating system because they require only a high level interface to communicate with the host (eg. transport commands in the case of the iSXM 552A board). The result is a powerful system building block which enables the OEM to network MULTIBUS I based systems with different operating systems. Applications for the 552A products include networked multiuser XENIX 286 based systems for the office and laboratory, iRMX-based systems for real-time applications, or many other system applications.



280385-1

September 1986
Order Number: 280385-001

THE iSBC® BOARD vs THE iSXM™ BOARD

The iSBC 552A version is a board that offers the hardware necessary for the user to construct an IEEE 802.3 front-end processor for custom requirements. The Intel iNA 960 ISO standard transport and network software can be configured and optimized to run on the iSBC 552A board.

The iSXM 552A version is a product that is preconfigured for Intel's family of System 310 products, includes the necessary internal system cabling, and is fully qualified to run in System 310 products. The iSXM 552A board supports the iNA 961 ISO standard transport and network software with no configuration activities required of the customer. iSXM 552A board customers receive the iNA 961 software through a separate purchase of a software license.

ARCHITECTURE DESCRIPTION

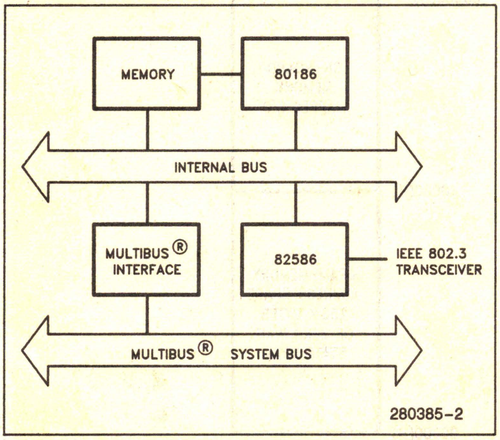


Figure 1. iSBC®/iSXM™ 552A Architecture

The iSBC and iSXM 552A boards consist of the following major architectural blocks (see Figure 1): an 80186 processor running at 8 MHz, the IEEE 802.3 I/O channel based on the 82586 LAN coprocessor, the on-board memory consisting of ROMs and 256K of zero wait state dynamic RAM, and the MULTIBUS I interface.

Processor

The iSBC 552A board contains an 80186 processor operating at 8 MHz. It is responsible for implement ing the intelligent interface between the iSBC 552A board and a host processor. The 80186 processor runs the iNA 960/961 transport software and delivers data between user buffers in MULTIBUS I memory and iNA 960/961 buffers on the iSBC and iSXM 552A boards. iNA 960/961 software is responsible for the reliable transfer of information across the IEEE 802.3 compatible network.

The 80186 and 82586 use both synchronous and asynchronous ready logic. The 80186 chip select lines are used to select memory mapped I/O locations.

The 80186 supplies the timers and the interrupt controller on the iSBC 552A board. The interrupt controller is used in the fully nested mode. The inputs and the outputs of the 80186 timers are not connected to external sources and destinations. Timer clocking and timer interrupts are generated internally in the 80186.

Memory

The iSBC/iSXM 552A board is equipped with 256K Bytes of zero wait state dynamic RAM and 16K Bytes of EPROM. The EPROM parts (Type 2764) are in two 28-pin sockets (JEDEC 27256 or 27572). The user can substitute parts (Type 27512) to provide 128K Bytes of EPROM.

The one megabyte address space of the 80186 is divided into four quadrants (see Figure 2). The first quadrant (0–256K Byte) is reserved for local EPROM memory and the last quadrant (768–1000K Byte) is reserved for local DRAM memory. The second quadrant (256–512K Byte) is used for memory mapped I/O. The iSBC/iSXM 552A board is totally memory mapped. The third quadrant (512–768K Byte) maps into a 256K Byte MULTIBUS I window. This window allows the iSBC/iSXM 552A board to access a total of 16M Byte of MULTIBUS I memory in 256K Byte segments. The iSBC/iSXM 552A board does not contain any memory which is accessible by other boards over the MULTIBUS I system bus.

The 256K Byte MULTIBUS I window starts on 64K Byte boundaries anywhere in the 16M Byte MULTIBUS I memory. The starting location of this window is determined by a memory mapped I/O latch described in the "iSBC 552A User Interface" section.

Memory mapped I/O locations are selected by the PCS and the MCS control lines of the 80186 processor. Functions controlled by memory mapped I/O are discussed in the "iSBC 552A User Interface" section.

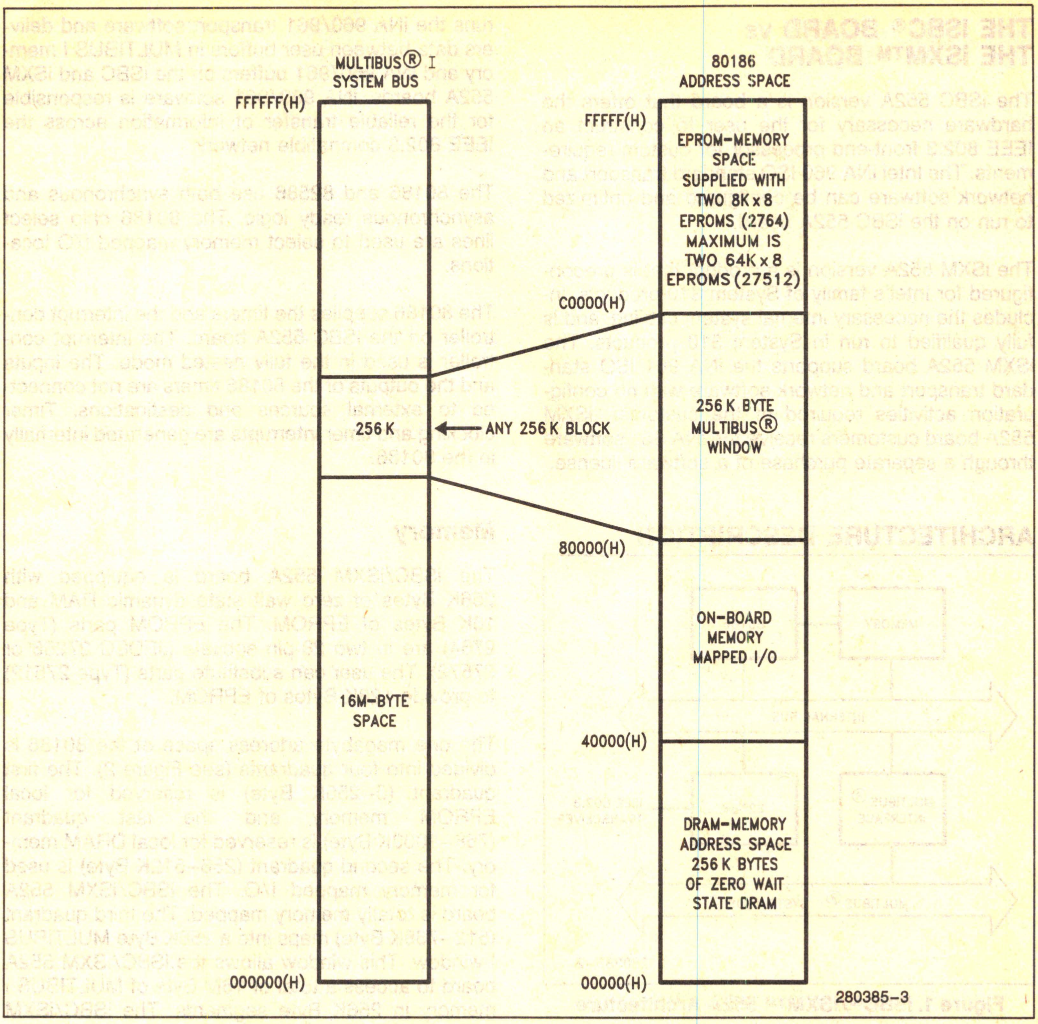


Figure 2. iSBC® iSXM™ 552A Memory Configuration

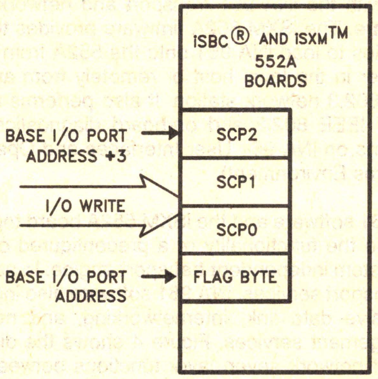
IEEE 802.3 Interface

The IEEE 802.3 Interface on the iSBC/iSXM 552A board is based on the 82586 LAN controller. Data is transferred between the on-board memory of the iSBC/iSXM 552A board and the 82586 controller by 82586 initiated DMA. The 82586 initiates the DMA cycles by activating the HOLD signal to the 80186 processor. The DMA cycle begins when the 80186 processor activates the HOLD ACKNOWLEDGE signal.

Each iSBC/iSXM 552A board is manufactured with a unique default 48-bit IEEE 802.3/Ethernet network address stored in an address PROM. This address PROM is protected by checksum and can be read by utilizing the on-board memory mapped I/O. The 82586 can be programmed to have this or any other Ethernet address.

MULTIBUS® I Interface

The iSBC/iSXM 552A board can access the MULTIBUS I with an 8- or 16- bit data path and can support up to 24-address bits. An I/O operation by the 80186 on the iSBC/iSXM 552A board normally accesses the I/O ports on the 80186 that controls the processor's interrupt controller and timers. MULTIBUS I/O is disabled in this normal operation. iSBC/iSXM 552A MULTIBUS I/O operations can be enabled or disabled by writing to memory mapped I/O control locations (Table 2). When the MULTIBUS I/O is enabled, the iSBC/iSXM 552A board can write or read the complete 64K Bytes of I/O space locations.



Base I/O port address = configurable. If 8-bit I/O is used, base port address is configurable from 0-OFCH. If 16-bit I/O is used, base port address is configurable from 0-OFFFCH.
Flag byte: see Table 1.
SCP0-SCP2: I/O written by host processor and read by 80186 on iSBC and iSXM 552A SCP0 can be jumper preset.

Figure 3. iSBC® 552A MULTIBUS® I Communication Interface

Table 1

Value Written to Flag Byte Port	Action
1	Resets iSBC 552A Board
2	Interrupts 80186 on Interrupt Level 1
4	Clears a MULTIBUS Interrupt Previously Generated by the iSBC 552A Board

A host processor in a system communicates with the iSBC/iSXM 552A board via a flag byte port and three other byte registers in the MULTIBUS interface. These registers are called the "System Configuration Pointer" registers (SCP0-SCP2). The flag byte port and the SCP registers are presented as 4 consecutive MULTIBUS I/O ports to the host processor. The locations of these I/O ports on the MULTIBUS are configurable on the iSBC 552A (Figure 3). To the 80186 processor on the iSBC/iSXM 552A board, the three SCP registers are memory mapped locations.

The flag byte port is used by the host processor to reset the iSBC/iSXM 552A board, to interrupt the 80186 processor, and to reset a MULTIBUS I interrupt generated by the iSBC/iSXM 552A board (Table 1). SCP0-SCP2 are general purpose registers that the host processor can I/O write to and the iSBC/iSXM 552A board can read from. SCP0 can also be preset by hardware jumpers.

iSBC® 552A FUNCTIONAL DESCRIPTION

The iSBC 552A board is a high performance general purpose IEEE 802.3 compatible COMMengine designed to offload a host processor in a system from transport layer and network layer communication processing. The board supports user written communications software for unique applications or it can run Intel's iNA 960/961 transport and network software in standard applications. When running iNA 960 software, the iSBC 552A board provides the host processor with reliable process to process message delivery. User messages to be sent are copied by iNA 960 software into iSBC 552A board local memory for transmission. Packets received from the network are first buffered and reassembled into messages on the iSBC 552A board. These received messages are then delivered to the user.

The iSBC 552A board makes use of the functions on the 82586 controller to implement a number of network functions. These functions include reprogramming the iSBC 552A station address, Multicast packet reception filtering, and loopback diagnostics. The 82586 also records a set of network statistics information. Information stored includes the number of CRC and alignment errors, the number of occurrences of no receive buffer resources and the number of DMA overruns/underruns.

The iSBC 552A can be configured to have a range of EPROM memory configurations up to 128K Bytes using 27512's.

The iSBC 552A board and iNA 960 software combination offers a flexible and configurable transport

COMMengine, and allows a user to optimally configure the system for highest performance. The iSXM 552A and iNA 961 combination offers a preconfigured turn-key solution. In both cases, iNA 960/961 software and the 552A significantly reduce the design cycle involved in designing and implementing a transport COMMengine.

For additional information about iNA 960/961, please refer to the iNA 960/961 data sheet.

iSBC® 552A User Interface

The iSBC 552A board communicates with a host processor through a handshake of interrupts. The host processor can generate flag byte interrupts to the 80186 on the iSBC 552A and the iSBC 552A can generate MULTIBUS I interrupts to the host processor. The host processor and the iSBC 552A board can also communicate through shared MULTIBUS I system memory. None of the on-board buffer on the iSBC 552A board is accessible to the host processor but the iSBC 552A can read and write all of the 16M Byte of MULTIBUS I system memory.

The host processor and the iSBC 552A board further communicate through the SCP registers. These byte registers can be I/O written by the host and can be read through memory mapped I/O by the iSBC 552A processor.

The 80186 processor controls the iSBC 552A through memory mapped I/O. Functions that are controlled are listed in Table 2.

OPERATING ENVIRONMENTS

The iSBC/iSXM 552A is designed to function in any MULTIBUS I system as a communications processor. It can function as both a MULTIBUS I bus master or a slave. As a MULTIBUS I master, it can access up to 16M Byte of host memory and 64K Byte of I/O address. As a MULTIBUS I slave, it occupies four consecutive I/O locations on the MULTIBUS I system memory. These locations are reserved for the flag byte and the three SCP registers.

iSXM™ 552A FUNCTIONAL DESCRIPTION

The iSXM 552A board is offered to operate specifically with the iNA 961 transport and network layer software. The iSXM 552A firmware provides the capabilities to load iNA 961 onto the 552A from either a buffer in the local host or remotely from another IEEE 802.3 network station. It also performs a variety of IEEE 802.3 and on-board diagnostics (see sections on iNA 961 User Interfaces and Operating Systems Environment).

iNA 961 software and the iSXM 552A board together provide the functionality of a preconfigured operating system independent transport engine. In addition to transport services, iNA 961 software also includes extensive data link, internetworking, and network management services. Figure 4 shows the distribution of network seven layer functions between iNA 961/iSXM 552A and the host processor. Table 3 shows some examples of functions provided by iNA 961. Refer to the iNA 960/961 data sheet for more iNA 961 information.

Table 2. iSBC® 552A Memory Mapped Functions

80186 Chip Select Lines	Read/Write by 80186	Functions
MCS	R	MULTIBUS I Interface registers (System Configuration Pointer Registers, see "MULTIBUS Interface" Section)
PCS	W	Channel Attention to 82586
	R	Reading iSBC 552A Ethernet Address PROMS
	W	Controlling Loopback of the Serial Interface
	W	Disabling and Enabling MULTIBUS I/O
	W	Generating and Clearing iSBC 552A Interrupts to the MULTIBUS System Bus
	W	Controlling the On-Board LED
	W	Latches the MULTIBUS Window Segment (8 most Significant Bits of 24-Bit Address)

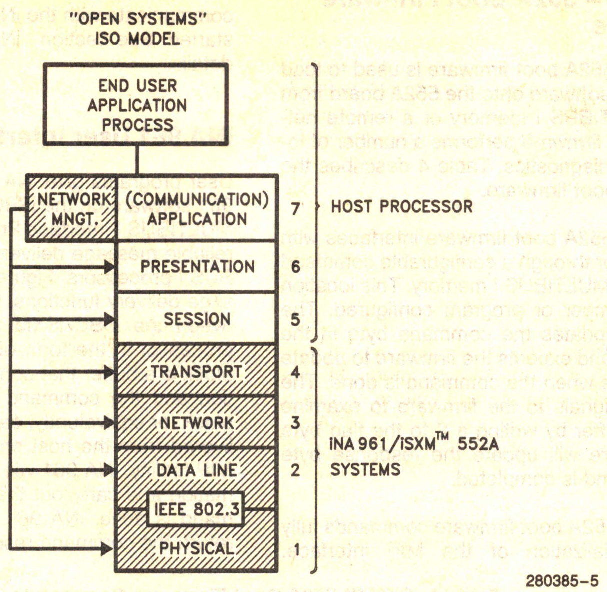


Figure 4. iNA 961 Configuration on iSXM™ 552A Board

Table 3. iNA 961 Services

Transport	Virtual Circuit Open: Establish a Virtual Circuit Database Send Connect: Actively Try to Establish a Virtual Connection Await Connect: Passively Awaits the Arrival of a Connection Request Send: Send a Message Receive: Post a Buffer to Receive a Message Close: Close a Virtual Circuit Datagram Send: Send a Datagram Message Receive: Post a Buffer to Receive a Datagram Message
Data Link	Transmit: Transmit a Data Link Packet Receive: Post a Buffer to Receive a Data Link Packet Connect: Make a Data Link Logical Connection (Link Service Access Point, IEEE802.3/802.2) Disconnect: Disconnect a Data Link Logical Connection Change Ethernet Address: Change the Ethernet Address Add Multicast Address: Add a Multicast Address Delete Multicast Address: Remove a Multicast Address Configure 82586: Configure the 82586 Controller
Network Management	Read/Clear/Set Network Objects (Local/Remote): Read/Clear/Set Local or Remote iNA 960 Network Parameters Read/Set Network Memory (Local/Remote) Read/Set Memory of the Local or a Remote Station Useful in Network Debug Process. Boot Consumer: Requests a Network Boot Server to Load a Boot File into this Station Echo: Echo a Packet between this Station and Another Remote Station on the Network

iSBC®/iSXM™ 552A Boot Firmware User Interface

The iSBC/iSXM 552A boot firmware is used to load iNA 961 or other software onto the 552A board from either local MULTIBUS I memory or a remote network station. The firmware performs a number of local and network diagnostics. Table 4 describes the functions of the boot firmware.

The iSBC/iSXM 552A boot firmware interfaces with the host processor through a configurable command buffer location in MULTIBUS I memory. This location can be either jumper or program configured. The host processor updates the command byte in the command buffer and expects the firmware to update the response byte when the command is done. The host processor signals to the firmware to examine this command buffer by writing a 2 to the flag byte port. The firmware will update the response byte when the command is completed.

The iSBC/iSXM 552A boot firmware commands fully support the initialization of the MIP interface. The MIP interface is used by the host processor to communicate with the iNA 961 once it is loaded and started. See section "iNA 961 User Interfaces" for details.

iNA 961 User Interfaces

User programs give iNA 960 commands to the iNA 961 software on the iSBC/iSXM 552A board via the MULTIBUS I Interface Protocol (MIP). MIP is an Intel reliable message delivery protocol between MULTIBUS I processors. Figure 5 illustrates how this message delivery functions. Commands are passed between the iSBC/iSXM 552A board and the host processor in the form of request blocks. A request block is a buffer that contains a command specification and the command parameters. Each request block (or equivalently, each command) is reliably delivered from the host processor to iNA 961 via the MIP facility. iNA 961 will extract the command information and carry out the command. After the command is done, iNA 961 will use the MIP facility to return the command result to the user program.

Table 4. iSXM™ 552A Boot Firmware Commands

Command	Function
Presence	This command will indicate that the boot firmware is functional by returning the version number of the firmware, the power on diagnostic result, and the default Ethernet address of the iSXM 552A board.
Load	Load a program from MULTIBUS memory into a designated location in the iSBC 552A memory.
Load and Go	Load a program from MULTIBUS bus memory into a designated location in the iSXM 552A memory. Proceed to start this program once it is loaded. This command also initializes the MIP interface on the iSXM 552A board.
Echo	Echo a packet between this iSXM 552A board and another station on the network.
Remote Boot	This command requests a remote boot server station to download software onto the iSXM 552A board.
MIP Initialize and Start	Used after a remote boot. This command initializes the MIP interface on the iSXM552A board and then start the software loaded by the remote boot command.

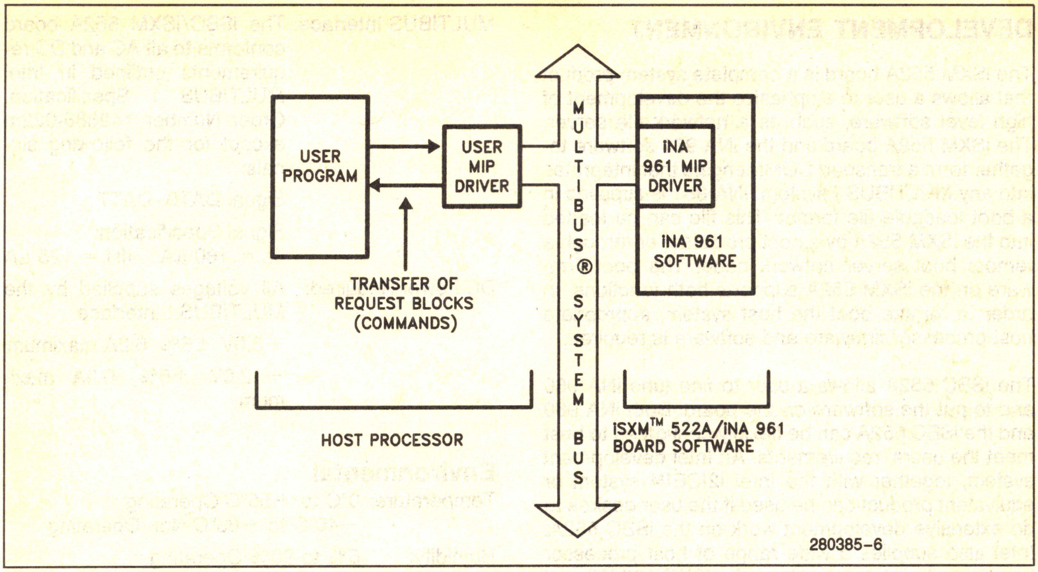


Figure 5. iNA 961 MIP Interface

iNA 961 request blocks are in the same formats as iNA 960 commands. Refer to the iNA 960/961 data sheet and reference manuals for more details on iNA 960/961 software.

Operating Systems Environment

The iSBC/iSXM 552A board and iNA 960/961 software can function in any MULTIBUS I environment. The communication between the iSBC/iSXM 552A and the host processor is entirely independent of any host operating systems. iNA 960/961 uses the MIP protocol to interface with the host processor. The MIP is a reliable, host operating system independent, process to process communication scheme between any processors on the MULTIBUS I System Bus. iNA 960/961 can service multiple processes utilizing its services at the same time.

A host processor passes iNA 960/961 commands and buffers in the MULTIBUS I system memory to the iNA 960/961 software. This software is responsible for updating the response fields of these commands. It is responsible for copying the user send buffer in MULTIBUS I system memory into its onboard buffers for transmission and for copying received messages to user buffers in MULTIBUS I system memory.

Diagnostics

The iSBC/iSXM 552A board offers a range of power up diagnostics designed to ensure that the 80186 processor, the memory, and the IEEE 802.3 interface are functioning properly. Table 5 describes these diagnostics.

Table 5. Functions Checked by iSXM™ 552A Diagnostics

1. Insufficient RAM
2. RAM March Pattern Test
3. Ram Ripple Data Test
4. Boot Firmware PROM Checksum
5. Address PROM Checksum
6. 80186 Interrupt Controller
7. 80186 Timer Controller
8. 82586 Initialization
9. 82586 CRC Check
10. 82586 Broadcast Packet Recognition
11. 82586 External Loopback
12. 82586 Individual Address Recognition
13. 82586 Multicast Address Recognition
14. 82586 Reset
15. 82586 Diagnose Check

DEVELOPMENT ENVIRONMENT

The iSXM 552A board is a complete system product that allows a user to emphasize the development of high level software, such as a network file server. The iSXM 552A board and the iNA 961 software together form a transport COMMengine that integrates into any MULTIBUS I system. iNA 961 is supplied in a boot loadable file format. This file can be loaded into the iSXM 552A by a host processor or through a remote boot server network node. The boot firmware on the iSXM 552A supports both functions. In order to remote boot the host system, appropriate host processor firmware and software is required.

The iSBC 552A allows a user to fine tune iNA 960 and to put the software on the board. Both iNA 960 and the iSBC 552A can be flexibly configured to best meet the users' requirements. An Intel development system, together with the Intel I2ICE™ system or equivalent product can be used if the user desires to do extensive development work on the iSBC 552A. Intel also supplies a wide range of host processor boards and systems (such as the iSBC 286/12 and system 310) that will function well both with the iSBC 552A or the iSXM 552A board.

SPECIFICATIONS

Data Transfer: 8 or 16 bits

Average Raw MULTIBUS I Transfer Rate:

8.7M bits/second (450 ns., 16-bit system memory and no MULTIBUS I contention)

Transceiver Interface

Transmit Data Rate: 10M bits/second

Signal Levels: Series 10,000 ECL-compatible

Host Interrupts: One MULTIBUS I non-vector interrupt for use in system/host handshaking

MULTIBUS Interface: The iSBC/iSXM 552A board conforms to all AC and DC requirements outlined in Intel MULTIBUS I Specification. Order Number 142686-022m except for the following signals:

Signal DAT0–DAT7

Signal Specification:
IIL = 180 μA IIH = 125 μA

DC Power Required: All voltages supplied by the MULTIBUS I interface

+5.0V ±5%, 6.2A maximum

+12.0V ±5%, 0.5A maximum

Environmental

Temperature: 0°C to +55°C Operating
−40°C to −65°C Non-Operating

Humidity: 5% to 90% Operating
5% to 95% Non-Operating

ORDERING INFORMATION

Part Number	Description
SBC552A	IEEE 802.3 COMMengine
SXM552A	IEEE 802.3 Transport Engine for iNA961 and SYP310 systems
iNA960	Configurable transport software usable with the SBC552A
iNA961	Preconfigured transport software for the SXM552A

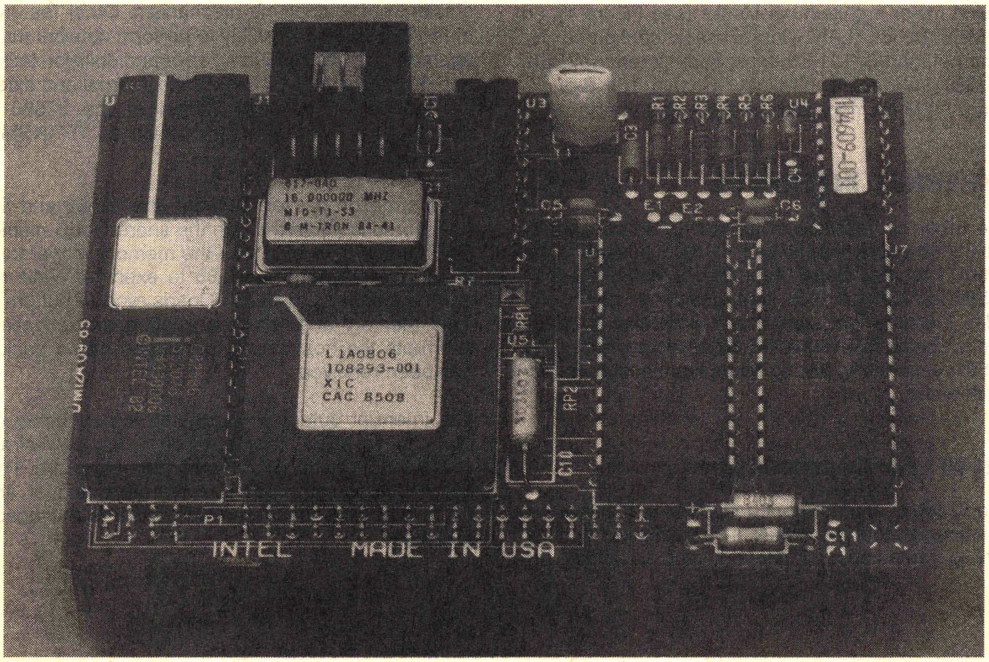
iSBX™ 586 ETHERNET DATA LINK ENGINE

Member of the OpenNET™ Product Family

- **Provides an IEEE 802.3 (Ethernet) Connection for Host Boards with 16-Bit iSBX™ Capabilities**
- **Based on Intel's 8 MHz 82586 LAN Coprocessor Chip which Includes the following features:**
  - **— Automatic Retransmission**
  - **— On-Board Multicast Address Filtering**
  - **— Host Interface via Buffer Chaining**
- **16 Kbytes of Local Dual-Ported Buffer RAM**
- **Single Wide iSBX™ MULTIMODULE™ that Conforms to Intel's iSBX Bus Specifications**
- **Compatible with iNA 960 ISO Transport Layer Software**
  - **— Direct Support for iRMX™ Operating Systems**
  - **— Source Code Support for Other Operating Systems**

The iSBX™ 586 Ethernet Data Link Engine is a single wide iSBX sized card that provides a low cost Ethernet controller MULTIMODULE™ for MULTIBUS® based systems with 16-bit iSBX bus capabilities. Based on the 82586 Local Area Network Coprocessor, the iSBX 586 implements the data link (Layer 2) and physical (Layer 1) layers of the International Standards Organization (ISO) Open Systems Interconnect (OSI) Reference Model. This allows the iSBX 586 to supply an IEEE 802.3 10 Mbps (Ethernet) connection for an iSBC board with iSBX capabilities.

The iSBX 586 MULTIMODULE is a low cost building block that can implement an Ethernet connection at various levels of integration. One application for the iSBX 586 is as a "best effort" datagram message delivery engine. In conjunction with the host iSBC board running iNA 960 R2.0 ISO Transport Software and iRMX Networking software, the iSBX 586 can allow for a four or a complete seven-layer, OpenNET compatible solution for Ethernet connections.



231668-1

November 1985
Order Number: 231668-002

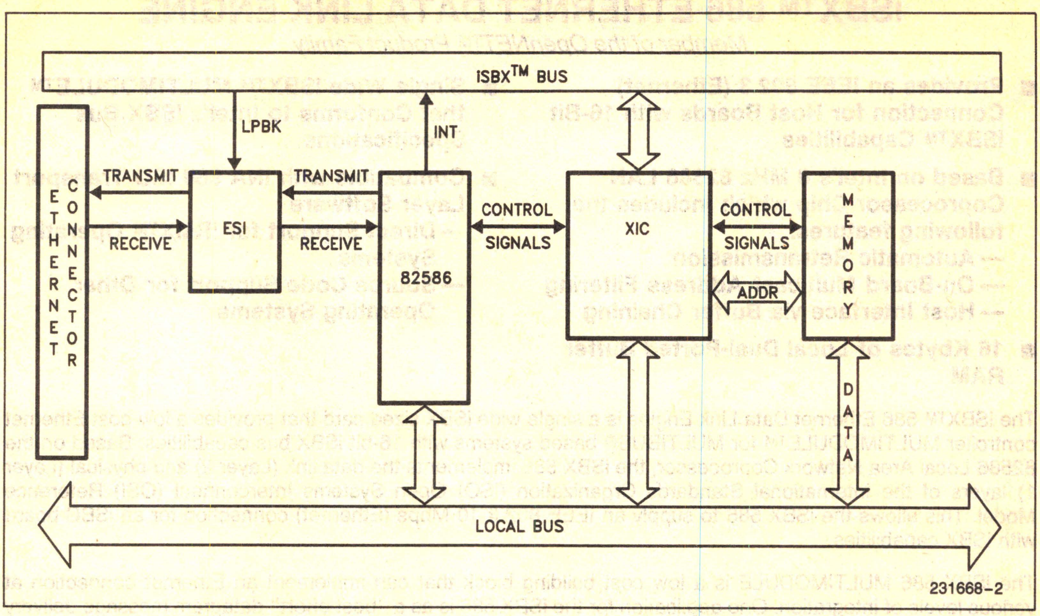


Figure 1. iSBX™ 586 Block Diagram

ARCHITECTURAL DESCRIPTION

The iSBX 586 MULTIMODULE consists of the following major architectural blocks (see Figure 1): an 8 MHz 82586 LAN Coprocessor configured for Ethernet, and 82501 Ethernet Serial Interface (ESI), an iSBX Interface Controller (XIC) chip, 16 KB of on-board RAM memory and the iSBX Bus Interface.

ETHERNET INTERFACE

The Ethernet Interface of the iSBX 586 consists of the 82586 Local Area Network Coprocessor and the 82501 Ethernet Serial Interface components. The 82586 is made up of a serial machine, which provides the data link control and a parallel interface that is compatible with MCS-86 based systems. The 82501, essentially, is a 10 Mbps Manchester encoder/decoder.

82586 LAN Coprocessor

The 82586 is an intelligent peripheral that completely manages the processes of transmitting and receiving frames over the network, thus off-loading the host CPU of communication management tasks. The 82586 features an on chip DMA controller which allows it to access the local memory through the efficient buffer chaining mechanism. Other features of the 82586 are the ability to perform network management activities including error and collision tallies and diagnostic capabilities via the internal and external loopback functions. Control of the 82586 is through high level commands such as TRANSMIT and CONFIGURE.

All information passed between the 82586 and the host board is made through the shared local memory. The host CPU may load the memory with a command and prompt the 82586 to execute. While receiving a packet, the 82586 loads receive buffers in local memory and, after completing the reception, interrupts the host board to indicate that a packet has been received.

The interrupt output of the 82586 is connected directly to the iSBX interface and is the only direct contact that the 82586 has with the host board. This interrupt is used to inform the host board of any event that has occurred which requires the host's attention.

A typical local bus cycle begins with a channel attention issued by the baseboard to the iSBX 586 (see section on the iSBX Bus Interface). Following the channel attention, the 82586 generates a HOLD. The iSBX Interface Controller (XIC) arbitrates the request, releases the control lines and issues a HOLDA (Hold Acknowledge). The 82586 can then proceed with normal read and write cycles. After completing the required memory accesses, the 82586 de-asserts the HOLD signal and the XIC removes the HOLDA. After completing the cycle, an interrupt to the host board is generated. For further information regarding the 82586, refer to the 82586 Data Sheet.

82501 Ethernet Serial Interface

The 82501 is a 10 Mbps Manchester encoder/decoder designed to work directly with the 82586 LAN Coprocessor. Additionally, the 82501 generates the 10 MHz transmit and receive clocks for the 82586 and drives the transceiver cable. The internal loopback function of the 82501 allows for fault isolation.

Loopback is asserted directly through the iSBX Interface Bus and as such, is controlled by the host CPU. When asserted, the 82501 routes the serial data through the transmit logic (without activating the output drivers) and back through the receive logic to be output to the 82586. For further information on the loopback function and for general 82501 information, refer to the 82501 Data Sheet.

iSBX™ Interface Controller

The iSBX Interface Controller (XIC) chip integrates the functions necessary to allow the 82586 LAN Coprocessor and the static memory on board the iSBX 586 to interface with the iSBX Interface Bus. The XIC chip was designed to accept all pertinent iSBX bus signals and act on them in accordance with the iSBX Bus Specification for 16-bit iSBX systems. The XIC chip is an Intel proprietary component and is not offered as a unique product.

The XIC arbitrates local bus control between the 82586 and the iSBX Bus Interface. After decoding the chip select, address and command lines from the iSBX Bus and the HOLDA signal from the 82586, the XIC synchronizes the request, determines priority and surrenders control of the local bus to the appropriate bus master. The 82586 has priority over the iSBX Interface Bus in terms of local bus arbitration. Once the arbitration has been resolved, the XIC chip is responsible for activating the proper address lines and chip selects for local memory. Additionally, the XIC turns on the proper data drivers and manages the memory control lines.

On Board Memory

The iSBX has 16 Kbytes of on board local RAM that serves as a communication liaison between the 82586 and the host CPU as well as providing buffers for packet storage prior to transmission and after reception. The RAM consists of two 8K × 8-bit CMOS static RAM chips configured as a two byte word to provide the full 16 bits of data. The RAM is addressed from 0 to 3FFFH locally but may be accessed at any 16K boundary (0, 4000H, 8000H, etc.) by the host board. In this way, the 82586 can access the fixed System Configuration Pointer (SCP) at memory location 3FF6H. Refer to the 82586 Data Sheet for information on the SCP.

A standard 32 × 8-bit PROM is used to contain the unique Ethernet station address. The station address is factory programmed and can only be accessed by the host board via the iSBX Bus.

iSBX™ Bus Interface

The iSBX Bus Interface is a major portion of the iSBX 586 MULTIMODULE. The XIC provides the interface between the iSBX Bus and the 82586 LAN Coprocessor and the local memory. The iSBX 586 is addressed as if it is an I/O slave on the iSBX Bus. There are four iSBX ports allocated for baseboard communication. The decoding of the ports is outlined in Table 1. MA0–MA2 are iSBX bus address lines.

Table 1. iSBX™ Bus I/O Slave Address Decode

MA2	MA1	MA0	Function	Read/Write
X	0	0	Memory Access	RD/WR
X	0	1	iSBX™ Address Load	WR Only
X	1	0	Station Address Read	RD Only
X	1	1	Channel Attention	WR Only, Data = X

X = don't care

NOTE:
As described in the iSBX Bus Specification, 16-bit iSBX base boards may connect ADR1–3 to the MULTIMODULE MA0–2 lines.

Due to the lack of addresses on the iSBX bus, the local iSBX 586 memory address must be set prior to the actual read or write operation over the iSBX Bus. The baseboard must first set up the appropriate address by executing an I/O write to the iSBX Address Load port 1 (MA1 = 0, MA0 = 1). The data written to port 1 is considered the iSBX memory address for the following iSBX memory access. The baseboard accesses the memory by addressing the iSBX Memory Access port 0 (MA1 = 0, MA0 = 0) for either a read or a write operation. The previously loaded memory address automatically increments allowing

for sequential memory access without reloading the iSBX address (port 1).

Channel attention is the signal used by the host CPU to prompt the 82586 into action. The baseboard issues a channel attention by simply writing to the iSBX port address 3 (MA1 = 1, MA0 = 1). In response, the XIC chip asserts the Channel Attention signal directly to the 82586.

The unique, factory programmed Ethernet station address can only be read by the host board. Reading the station address is accomplished by the base board issuing an I/O read to the iSBX port address 2 (MA1 = 1, MA0 = 0). The PROM address space is between 0 and 3EH.

A typical iSBX Bus Cycle is intiated by the baseboard activating the appropriate address, chip select and command lines. After the XIC chip receives the active address and chip select signals, it issues an $\overline{\text{MWAIT}}$ to the baseboard. When the command is received, the XIC arbitrates between the 82586 and the baseboard. If the arbitration is resolved in favor of the baseboard, the XIC turns on all drivers on the iSBX 586 board for the current cycle. Subsequently, the $\overline{\text{MWAIT}}$ signal is de-asserted, allowing the baseboard to complete the cycle.

OPERATING ENVIRONMENTS

The iSBX 586 is designed to operate as a slave to MULTIBUS hosts with 16-bit iSBX bus capabilities. Because the iSBX 586 has no processing ability, all associated software must be executed by the host. Most of the functions of the Data Link and Physical layers of the ISO Model are supported by the iSBX 586. iNA 960 R2.0 is Intel's ISO compatible software package for the Network and Transport Layers. For the upper layers, iRMX Networking Software can be used directly in conjunction with iNA 960 and the iSBX 586. Other operating systems can be supported through purchase of the iNA 960 source code (see Figure 2).

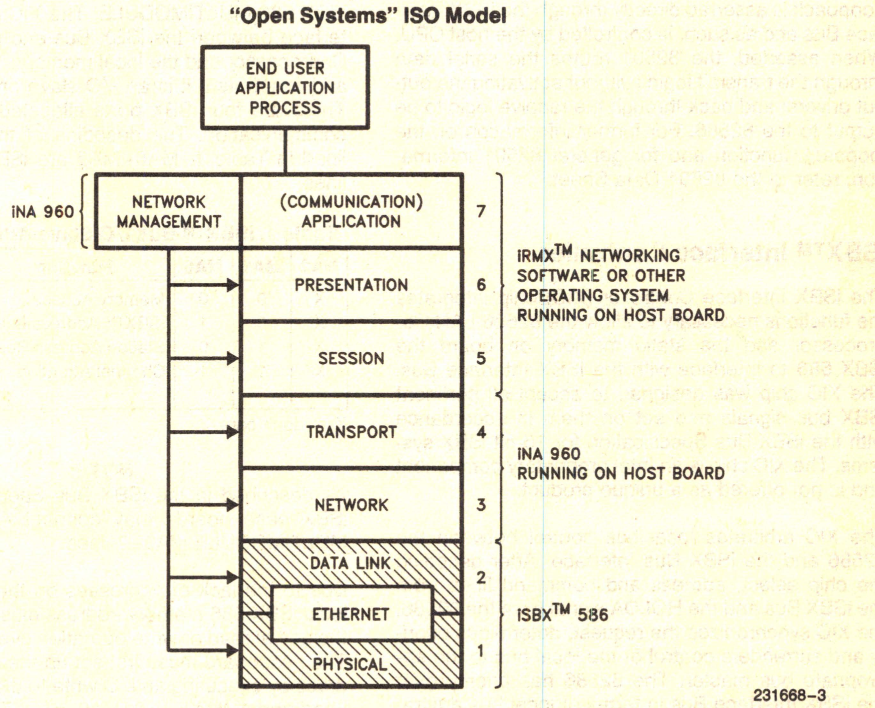


Figure 2. Mapping the iSBX™ 586 into the ISO Model

ORDERING INFORMATION

Part Number	Description
iSBX 586	Ethernet Data Link Engine

SPECIFICATIONS

iSBX Interface

Data Transfer	16 bits
Signal Levels	See the iSBX 586 Hardware Reference Manual
Signals Supported	All iSBX bus signals are supported except: MA2, MINTR1, MCLK, OPT1, MDACK, TDMA, MDRQT, −12V
Serial Interface	IEEE 802.3 compatible
DC Power Requirements	All voltages supplied by the iSBX Interface +5V DC ±5%, 2A max. +12V DC ±5%, 1A max.

ENVIRONMENTAL

Temperature	0°C to 55°C Operating (Free moving air across the base board and iSBX 586) −40°C to +65°C Non-Operating
Humidity	5% to 90% Operating 5% to 95% Non-Operating

Refer to the iSBX™ 586 Hardware Reference Manual (not supplied) for details.

iSBC® 554
MAP COMMUNICATIONS ENGINE

- Provides IEEE 802.4 Networking Capability for MULTIBUS® Based Systems Running Under any Operating System
- Serves as a Complete Front End Communication Engine With the Capacity to Provide MAP Layers 1 Through 7 Capability for MULTIBUS® Based Hosts
- Runs on Board Intel's Proven iNA 960 Rel 2.0 Providing the ISO 8073 Transport Software and ISO 8473 Network Software as Required by the Map Specifications
- Runs on Board Intel's MAP-NET™ Software for Layers 5–7 of the Map Protocol
- Preconfigured Software Available for Seven Layer Map Engine, Four Layer Transport Engine or IEEE 802.4 to IEEE 802.3 Router
- 8 MHz 80186 Processor
- 256K Bytes of RAM of Which 128K Bytes Provide Dual Port Window Support
- 10 Mbps IEEE 802.4/Token Bus Modem Interface
- Sockets for up to 4 JEDEC 28 Pin Memory Devices, up to Maximum of 160K Bytes EPROM Storage
- One iSBX™ Bus Connector for I/O Expansion Capability
- Can Be Configured as Either a Master or a Slave in MULTIBUS
- On Board Diagnostic and Boot Firmware
- Available in Three Different Modem Frequencies/Channel Pairs

The iSBC 554 COMMengine product is designed to fit into front end LAN Communication processor applications. It allows the connection of MULTIBUS I based systems onto a MAP/IEEE 802.4 (Token Bus) LAN. COMMengines are dedicated communication processor boards. They allow the host processor board to off-load LAN communication related tasks onto the front end COMMengine. Therefore the host has more processing capability for user applications or other tasks. COMMengines also allow the networking of existing systems without forcing a redesign of the entire system architecture.

The iSBC 554 board can be used as a front end COMMengine for a MULTIBUS-based host running any operating system. This is because the on board software provides a high level interface to the host (e.g., application or transport level commands). This results in a powerful system building block which enables an OEM to connect MULTIBUS-based systems onto IEEE 802.4 10 Mbps LANs. Applications for the iSBC 554 include networked iRMX™-based systems for real time applications and networked XENIX* systems for laboratory and data base application. The iSBC 554 is preconfigured to run iNA 961 R2.0 transport and network software. iNA 961 R2.0 is a preconfigured version for the iSBC 554 of Intel's iNA 960 LAN software which implements the ISO 8073 Class 4 transport protocol and the ISO 8473 network layer protocol.

The iSBC 554 COMMengine supports multiple datalinks via the iSBX connector located on the iSBC 554 baseboard. The user has the option to interface any of Intels iSBX communication interfaces to support a two way router. For example iNA 960 supports the MAP/TOP router using the iSBX 586 interface. The preconfigured router software is supplied in iNA 961.

The iSBC 554 is also capable of running on a board MAP2.1SXMSW preconfigured implementation of the MAP software for layers 3 through 7 of the ISO/OSI model. This is an ideal turnkey solution for OEMs requiring a 7 layer MAP COMMengine. MAP-NET™ provides layers 5 through 7 of the MAP specifications and can be configured with iNA 960 R2.0 to run on the iSBC 554, providing a complete on-board seven layer COMMengine.

*XENIX is a trademark of Microsoft Corporation.

September 1986
Order Number: 231594-002

iSBC® 554 FUNCTIONAL DESCRIPTION

The iSBC 554 board is a preconfigured MAP Communication Engine with boot firmware and 256K bytes of RAM. The iSBC 554 board is offered for use with Intel's MAP-NET/iNA 960 based MAP software. The iSBC 554 firmware provides the capabilities to load Intel's MAP software on the iSBC 554 from either a buffer in the local host or remotely from another Token Bus station. It also performs a variety of on-board diagnostics.

The MAP-NET with iNA 960 R2.0 software and the iSBC 554 board together provide the functionality of a preconfigured OS independent 7 layer engine. In addition to transport services, iNA 960 R2.0 software also includes ISO 8473 Internet network layer, extensive data link and network management facility services. Figure 1 shows the configuration of MAP-NET and iNA 960 R2.0. Table 1 shows some examples of functions provided by MAP-NET and iNA 960 R2.0. iNA 961 R2.0 is a preconfigured version of iNA 960 for the iSBC 554. Refer to the iNA 960 R2.0 data sheet for more information.

MAP-NET is Intel's implementation of the MAP software for layers 5 through 7. Refer to the MAP-NET data sheet for more information. This implementation of layers 5 through 7 will run on the iSBC 554 along with iNA 960 R2.0. The iSBC 554 coupled with the software packages provides a high performance, 7-layer communication engine (see Figure 1). MAP 2.1SXMSW is also available as a preconfigured software package providing layers 3 through 7 of the MAP software. This package and the iSBC 554 provides a 7 layer turnkey MAP solution.

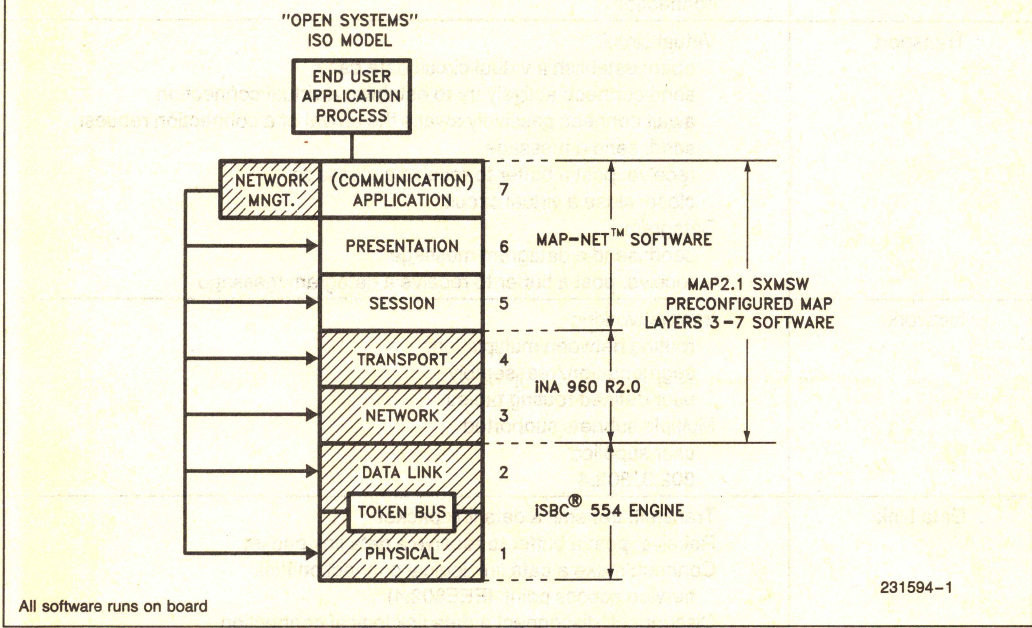


Figure 1. MAP-NET™ and iNA 960 Configuration on iSBC® 554 Board

Table 1. MAP-NET™ and INA 960 R2.0 Services

Application	File Transfer, Access and Management (FTAM) Provides remote operations on files (Create, Read, Write, Delete, Get File Attributes) Common Application Service Elements (CASE) Supports all the services provided by the lower ISO layers Provides name to address translation support Directory Services Performs name to address conversion Maintains local cache of resolved names Two forms of directory service—client service agent for local data base and directory-service agent for remote (master) data base
Session	Implements subset of ISO Session 8327 specified by the MAP 2.1 Specifications Provides "Graceful Close" "Graceful Close" allows the closing of a connection without any loss of queued requests It enhances the transport provided "close" which aborts a connection
Transport	Virtual circuit open: establish a virtual circuit data base send connect: actively try to establish a virtual connection await connect: passively awaits the arrival of a connection request send: send a message receive: post a buffer to receive a message close: close a virtual circuit Datagram send: send a datagram message receive: post a buffer to receive a datagram message
Network	Internetworking routing between multiple lans segmentation/reassembly user defined routing tables Multiple subnets supported user supplied 802.3, 802.4
Data Link	Transmit: transmit a data link packet Receive: post a buffer to receive a data link packet Connect: make a data link logical connection (link service access point. IEEE802.4) Disconnect: disconnect a data link logical connection Change token bus address Add multicast address Delete multicast address Configure TBH

Table 1. MAP-NET™ and iNA 960 R2.0 Services (Continued)

Network Management	Read/Clear/Set network objects (local/remote): read/clear/set local or remote MAPNET/iNA 960 network parameters Read/Set network memory (local/remote): read/set memory of the local or a remote station Useful in network debug process Boot consumer: requests a network boot server to load a boot file into this station Echo: Echo a packet between this station and another remote station on the network

ARCHITECTURE DESCRIPTION

The iSBC 554 board consists of the following major architectural blocks (see Figure 2): an 80186 processor running at 8 MHz, the Token Bus channel based on the Token Bus Handler chip set and the Token Bus Modem, the on-board memory consisting of ROM and RAM, the iSBX interface, and the MULTIBUS interface.

PROCESSOR

The iSBC 554 board contains an 80186 processor operating at 8 MHz. It is responsible for implementing the intelligent interface between the iSBC 554 board and a host processor. The 80186 processor runs the MAP-NET/iNA 960 R2.0 transport software and the data link software needed by the Token Bus Handler chip set. It is responsible for the delivery of data between user buffers in MULTIBUS memory and iNA buffers on the iSBC 554 board. The iNA software is responsible for the reliable transfer of information across the Token Bus LAN.

MEMORY

The one megabyte address space of the 80186 is divided into four quadrants (see Figure 3). The first quadrant (0–256K Byte) is local RAM memory. The second quadrant is memory mapped Token Bus Handler address. The third quadrant (512–768K Byte) maps into two MULTIBUS windows (128K Byte each). These windows allow the iSBC 554 board to access the total 16M Byte of MULTIBUS memory in 128K Byte segments. The fourth quadrant (768–1M Byte) is local ROM which contains the 80186 firmware, the Token Bus station address, and relocated 80186 internal registers.

The two 128K Byte MULTIBUS windows each start on 64K Byte boundaries anywhere in the 16M Byte MULTIBUS memory. The starting location of either window is determined by writing to a local I/O mapped latch.

Options on the iSBC 554 board allow up to 128K Byte of RAM to be accessible by the host. This dual port RAM is jumper selectable to appear anywhere in the MULTIBUS 16M Byte memory space on 128K Byte boundaries. The dual port RAM memory is a data link between the on board 80186, the token bus controller, and the bus master (if any) managing the systems functions. This shared dual port RAM can be used to transfer command, status and data between the on board 80186 processor and the host. This feature minimizes the necessity for the 80186 to access MULTIBUS while acquiring shared information. This has a direct positive effect on performance, serving to eliminate bus contention.

TOKEN BUS INTERFACE

The Token Bus interface on the iSBC 554 is implemented by the Token Bus Handler (TBH) chip set and the Token Bus Modem (TBM). Data is transferred between the on-board memory and the TBH by the TBH initiated DMA. The TBH will then pass data, operating according to the IEEE 802.4 Token Bus Specification, to the TBM which handles the physical interface to the Token Bus.

Each iSBC 554 board is manufactured with a unique default Token Bus network address stored in an address PROM. This address PROM is protected by checksum and can be read by utilizing the on board I/O.

MULTIBUS® INTERFACE

The iSBC 554 board can access the MULTIBUS with an 8- or 16-bit data path and can support up to 24 address bits. The internal 80186 registers are relocated into the local memory map to avoid conflicts with MULTIBUS I/O during 80186 internal register accesses. The iSBC 554 board is capable of accessing the MULTIBUS I/O from 384-64K (180H–FFFFH) Byte of I/O space locations.

A host processor in a system communicates with the iSBC 554 board via a flag byte port in the MULTIBUS interface. The flag byte port is presented as a

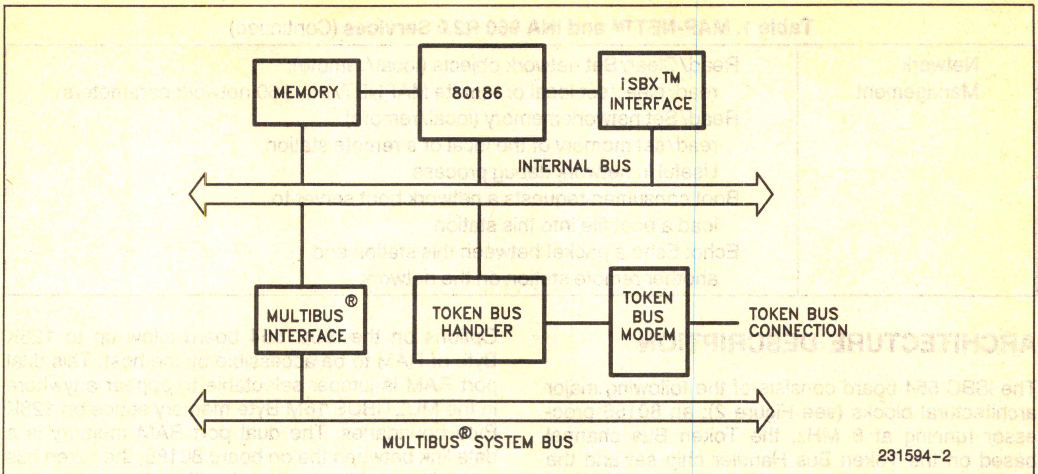


Figure 2. iSBC® 554 Architectural Blocks

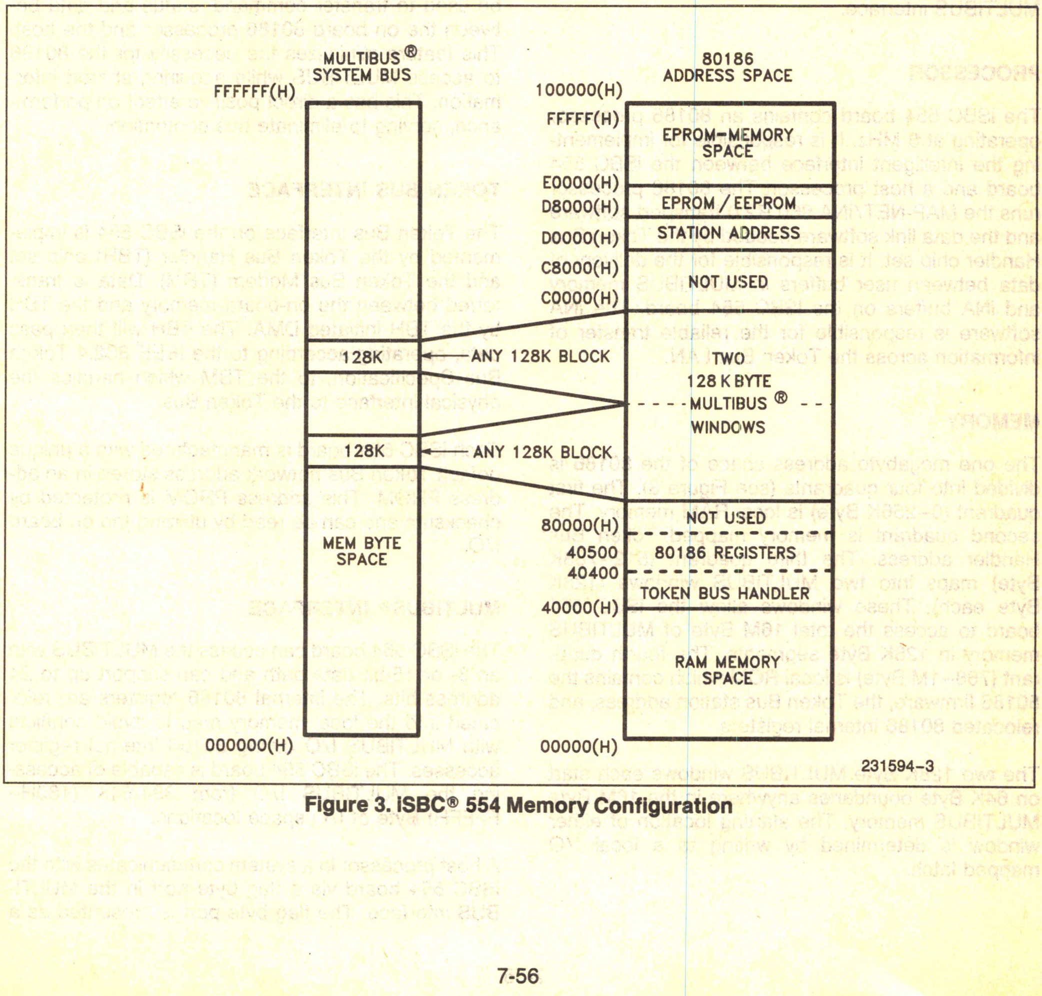


Figure 3. iSBC® 554 Memory Configuration

MULTIBUS I/O port to the host processor. The location of this I/O port on the MULTIBUS is configurable on the iSBC 554 board. To the 80186 processor on the iSBC 554 board, the flag byte is in a local I/O mapped location.

The flag byte port is used by the host processor to reset the iSBC 554 board, to interrupt the 80186 processor and to reset a MULTIBUS interrupt generated by the iSBC 554 board. The iSBC 554 board uses the flag byte to set or clear an interrupt to the MULTIBUS, or clear an interrupt from the MULTIBUS (Table 2).

For those applications requiring processing capacity and the benefits of multiprocessing (i.e., several CPUs and/or controllers logically sharing system tasks through the communication of the system bus), the iSBC 554 board provides full MULTIBUS arbitration control logic.

iSBX™ INTERFACE

One 8/16 bit iSBX MULTIMODULE™ connector is provided on the iSBC 554 board. Through this connector, additional on-board I/O functions may be added. iSBX MULTIMODULE boards optimally support functions provided by VLSI peripheral components such as additional parallel and serial I/O, analog I/O, small mass storage device controllers (e.g., cassettes and floppy disks) and other custom interfaces to meet specific needs. By mounting directly on the iSBC 554 board, less interface logic, less power, simpler packaging, higher performance, and lower cost results when compared to other alternatives such as MULTIBUS form factor compatible boards. The iSBX connector on the iSBC 554 board provides all signals necessary to interface to the local on-board bus, including 16 data lines for maximum data transfer rates. iSBX MULTIMODULE boards designed with 8-bit data paths and using the 8-bit iSBX connector are also supported on the iSBC 554 board. A broad range of iSBX MULTIMODULE options are available in this family from Intel. Custom iSBX modules may also be designed for use on the iSBC 554 boards. An iSBX bus interface specification and iSBX connector documentation are available from Intel.

iSBC® 554 USER INTERFACE

The iSBC 554 board communicates with a host processor through a handshake of interrupts. The host processor can generate flag byte interrupts to the 80186 on the iSBC 554. The iSBC 554 can generate MULTIBUS interrupts to the host processor. The host processor and the iSBC 554 can also com-

Table 2. Flag Byte Ports

Value Written to Flag Byte Port	Source	Actions
1	iSBC 554 board	Clears interrupt to the MULTIBUS
	MULTIBUS backplane	Resets iSBC 554 board
2	iSBC 554 board	Sets interrupt to the MULTIBUS
	MULTIBUS backplane	Sets interrupt to the iSBC 554 board
3	iSBC 554 board	Clears interrupt to the iSBC 554 board
	MULTIBUS backplane	Clears interrupt to the MULTIBUS

municate through shared MULTIBUS system memory. As much as 128K byte of the on-board RAM on the iSBC 554 board is accessible to the host processor and the iSBC 554 board can read and write all of the 16M byte of MULTIBUS system memory.

OPERATING ENVIRONMENTS

The iSBC 554 is designed to function in any MULTIBUS system as a communication processor. It can function as both a MULTIBUS bus master or a slave. As a MULTIBUS master, it can access up to 16M Byte of host memory and 64K byte of I/O address. As a MULTIBUS slave, it occupies one location reserved for the flag byte.

MAP-NET/iNA 960 R2.0 USER INTERFACES

User programs give MAP-NET/iNA 960 commands to the MAP-NET/iNA 960 R2.0 software on the iSBC 554 board via the MULTIBUS Interface Protocol (MIP). MIP is an Intel reliable process to process message delivery protocol between MULTIBUS processors. An implementation of the MIP protocol is provided on the iSBC 554 board for communication with the host. The corresponding MIP protocol implementation will have to be provided by the user on the host side for communicating with the iSBC 554. Figure 4 illustrates how this message delivery functions. Commands are passed between the iSBC 554 and the host processor in the form of request blocks. A request block is a buffer that contains a command specification and the command parameters. Each request block (or equivalently, each command) is reliably delivered from the host processor to MAP-NET/iNA 960 R2.0 via the MIP facility. MAP-NET/iNA 960 R2.0 will extract the command information and carry out the command. After a command is done, MAPNET/iNA 960 R2.0 will use the MIP facility to return the command result to the user program.

iNA 960 R2.0 request blocks are in the same formats as iNA 960 commands. Refer to the iNA 960 and MAP-NET data sheets and reference manuals for more details on the iNA 960 R2.0 and MAP-NET software.

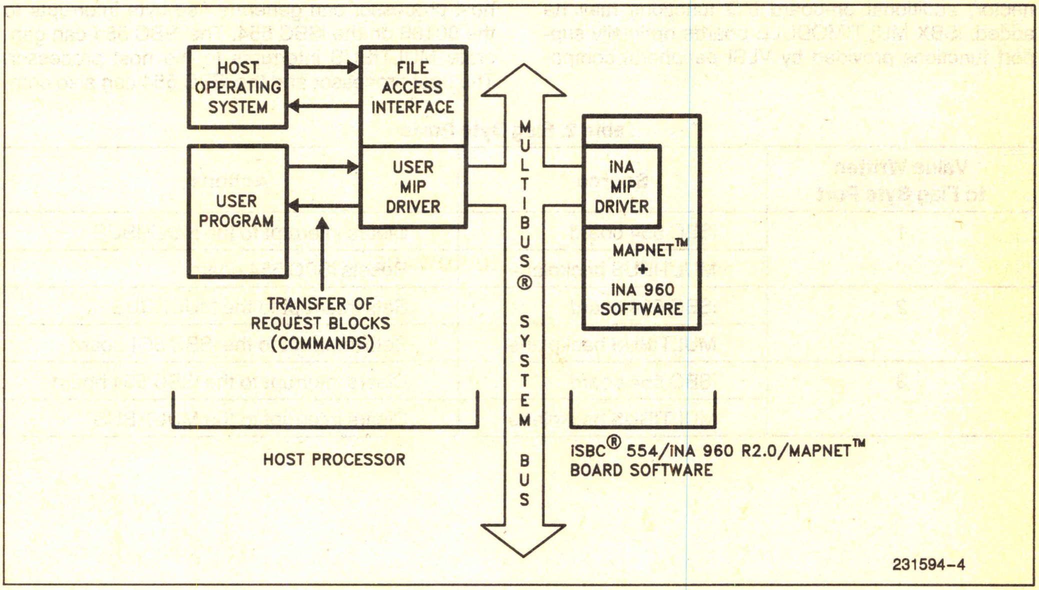


Figure 4. MAP-NET™ and iNA 960 MIP Interface

PRECONFIGURED SOFTWARE—iNA 961 AND MAP 2.1SXMSW

Preconfigured iNA 960 software supports the iSBC 554 COMMengine as a router (MAP/TOP) or as a transport and network communications engine. The iNA 961 package contains the iRMX 86 device driver, user interface utilities and preconfigured communications software.

MAP 2.1SXMSW preconfigured 7 layer solution supports all seven layers on the iSBC 554 COMMengine board. The layers that are located on the COMMengine includes FTAM, Directory Services, CASE, Session, Transport, Network, and the Data Link layer. In order to allow the maximum flexibility in interfacing users applications, the Network Management facility has been added. The combined layer solution provides the user with a certified, and conformance tested COMMengine, with the flexibility to modify all the system parameters.

The above preconfigured MAP product is supplied with iRMX 86 device drivers, user interface utilities and the 7 layer conformance tested software. The iSBC 554 COMMengine and software is designed to support generic operating systems and different host processors.

OPERATING SYSTEMS ENVIRONMENT

The iSBC 554 board and iNA 960 R2.0 software can function in any MULTIBUS environment. The communication between the iSBC 554 board and the host processor is entirely independent of any host operation systems. MAP-NET/iNA 960 R2.0 use the MIP protocol to interface with the host processor. MAP-NET/iNA 960 R2.0 can service multiple processes utilizing its services at the same time.

A host processor passes MAP-NET/iNA 960 R2.0 commands and buffers in the MULTIBUS system memory to the MAP-NET/iNA 960 R2.0 software. MAP-NET/iNA 960 R2.0 is responsible for updating the response fields of these commands. It is responsible for copying the user send buffer in MULTIBUS system memory into its on board buffers for transmission and for copying received messages to user buffers in MULTIBUS system memory.

iSBC® BOOT FIRMWARE USER INTERFACE

The iSBC 554 boot firmware is used to load MAP-NET/iNA 960 R2.0 or other software onto the 554 from either local MULTIBUS memory or a remote network station. The firmware performs a number of local and network diagnostics.

The iSBC 554 boot firmware commands fully support the initialization of the MIP interface. The MIP interface is used by the host processor to communicate with the iNA 960 R2.0 once it is loaded and started.

DIAGNOSTICS

The iSBC 554 board offers a range of power up diagnostics designed to ensure that the 80186 processor, the memory (EPROM and RAM), and the Token Bus Interface are functioning properly.

Available Literature:

— iNA 960 Release 2.0 Programmers Reference Manual

— iNA 960 Release 2.0 Configuration Guide

— iSBC 186/51 Hardware Reference Manual

— iSBC/iSXM 552 Hardware Reference Manual

— iSBC/iSXM 552A Hardware Reference Manual

— iSBC 554 Hardware Reference Manual

— MAP-NET Programmers Reference Manual

— RMX-NET Programmers Reference Manual

ORDERING INFORMATION

HARDWARE

Part Number	Modem Frequencies/Channel Pairs
iSBC 554-1	Transmit: 59.75 to 71.75 MHz/Ch. 3 and 4
	Receive: 252 to 264 MHz/Ch. P and Q
iSBC 554-2	Transmit: 71.75 to 83.75 MHz/Ch. 4A and 5
	Receive: 264 to 276 MHz/Ch. R and S
iSBC 554-3	Transmit: 83.75 to 95.75 MHz/Ch. 6 and FM1
	Receive: 276 to 288 MHz/Ch. T and U

SOFTWARE

Code	Description
MAP21SXMSWRO	License for preconfigured MAP 2.1 Layers 3–7 software.
MAP21SXMSWRF	Incorporation fee for preconfigured MAP 2.1 Layers 3–7 software (licence required).
MAPNET 21RO	License for configurable MAP 2.1 layers 5–7 software.
MAPNET 21RF	Incorporation fee for configurable MAP 2.1 layers 5–7 software (license required).
iNA 961 R2	Preconfigured transport and internet software for an IEEE 802.3 to IEEE 802.4 Router.
iNA 960 R2	Configurable MAP 2.1 layers 3–4 software.

HARDWARE/SOFTWARE PACKAGES

Code	Description
MAP554NODEKIT-X (X = 1, 2 or 3)	Package consists of one iSBC 554-X (X = 1, 2 or 3) and one MAP21SXMSWRF. This kit requires the prior purchase of MAP21SXMSWRO—the software license.

SPECIFICATIONS

Network Interface

Compatibility/ Conformance	IEEE 802.4, Token Bus 10 Mbps Broadband
Cable Connection	75Ω Output on Type F Female Connector
Head End	Operates with Remodulator Head End

Host Interface

MULTIBUS® Interface	Conforms to All AC and DC Requirements of the Intel MULTIBUS Specification
DC Power Required (Maximum Excluding iSBX)	+5 VDC – 5.5A +12 VDC – 0.3A –12 VDC – 0.15A

Environmental

Temperature:	0°C to 60°C Operating –40°C to +85°C Storage
Humidity:	5% to 95%, Non-Condensing, for Both Operating and Storage

MAP-NET™ COMMUNICATIONS SOFTWARE (MAPNET2.1 AND MAP2.1SXMSW) MEMBER OF THE OpenNET™ PRODUCT FAMILY

- **Supported by OpenNET™-Map Hardware and Software:**
  — iSBC®554 Token Bus Board
  — iNA 960 Communication Software
- **MAPNET2.1 Implements ISO/OSI Layers 5–7, as Specified by Map Version 2.1**
- **Designed to Interface with iNA 960 Rel 2.0—Intel's Transport and Network Software for Layers 3–4.**
- **MAPNET2.1, iNA 960 Rel 2.0 and the iSBC®554 Map Board Provide a Seven Layer, Modular and Configurable MAP Solution Based on Intel's OpenNET Architecture.**
- **Provides MAP 2.1 ISO FTAM, Session, CASE, Network Management/Directory Services**
- **Pre-Configured to Run on Intel's iSBC 554 MAP Board**
- **Preconfigured Software Provides Layers 3–7 of the MAP 2.1 Specifications.**
- **Preconfigured Map Software with the iSBC 554 Board Provides a Seven Layer Turnkey Solution.**

MAPNET2.1, iNA 960 Rel 2.0 and the iSBC 554 Map Board are ready-to-use building blocks for OEM suppliers of networked systems to implement ISO/OSI layers 1–7, as specified by MAP version 2.1. The Intel iSBC 554 board provides the data link and the IEEE 802.4 based physical layer for MULTIBUS® based systems. MAP-NET is designed to use the services and interface provided by Intel's iNA 960 Rel 2.0 Software package. iNA 960 Rel 2.0 provides the ISO 8473 network and ISO 8073 transport layers 3 and 4 of MAP 2.1. MAPNET2.1 provides layers 5–7 of the MAP 2.1 specifications and is designed to run on top of iNA 960 Rel 2.0 on the iSBC 554 board. Together the board and software modules provide a complete, seven layer configurable MAP solution for OEM's. The MAP-NET software is also available preconfigured with iNA 960 Rel 2.0 to run on the Intel iSBC 554 board. This preconfigured software with the board provides a complete 7 layer turnkey solution for MAP 2.1.

Figure 1. below indicates how Intel's OpenNET/MAP software and hardware products fit in the ISO/OSI reference model for MAP.

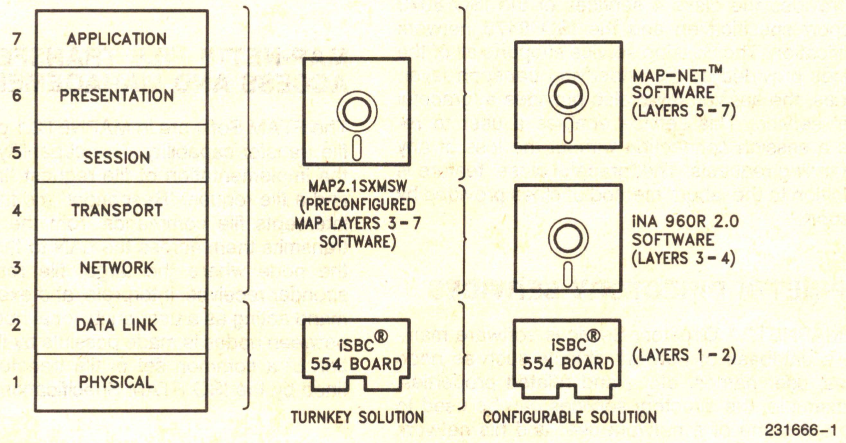


Figure 1. ISO/OSI Reference Model

September 1986
Order Number: 231666-002

FUNCTIONAL OVERVIEW

The Intel MAPNET2.1 software provides the following services specified by MAP 2.1; the session service, directory services, network management, FTAM and CASE. These services fit into the upper 3 layers of the ISO/OSI 7 layer model.

Using the Services of MAP-NET, users can initiate communications with other users on a MAP LAN, access information regarding resources available on a LAN, transfer files across a LAN and address other users on the LAN by logical names rather than numbered addresses.

MAP-NET is designed to interface with iNA 960 Rel 2.0. iNA 960 Rel 2.0 provides the network and transport protocol that is required by the map specification. Please refer to the iNA 960 data sheet for more information. The configurable software packages MAP-NET and iNA 960 Rel 2.0 are designed to run on the iSBC 554 for a complete, on-board, seven layer map COMMengine.

MAP2.1SXMSW is a preconfigured software package that incorporates the functions of MAP-NET and iNA 960 Rel 2.0. This package provides layers 3–7 and is designed to run on the iSBC 554 for a complete, on-board, seven layer, turnkey MAP COMMengine.

MAP-NET™ SESSION SERVICES

The MAPNET2.1 software implementation provides the Session services specified in the MAP version 2.1 specification. The session service is built on top of the iNA 960 Rel 2.0 transport service. iNA 960 Rel 2.0 provides the class 4 services of the ISO 8073 transport specification and the ISO 8473 network specification. The Session service supports all of the services provided by the underlying transport layer. Besides, the session layer also provides a 'graceful close' service. This service enables a user to release a session connection without the loss of any outstanding requests. The 'graceful close' feature is in addition to the 'abort' method of close provided by transport.

MAP-NET™ DIRECTORY SERVICES

The MAPNET2.1 Directory Services software maintains a database of network objects such as node names, user names, etc..., and related properties. For example, the directory services can be used to store the name of a network user and his network addresses as the properties associated with his name. A network user or application can query the directory service to retrieve information from this database. Users can also add or delete objects and properties from this database.

The Directory Services provided in MAPNET2.1 does the following:

— Runs on top of CASE

— Performs name to address conversion

— Maintains a local cache of resolved names

— Provides two forms of Directory Service—Client Service Agent for Local Data Base and Directory Service Agent for Remote/Master Data Base. (Can be configured to utilize the host memory pool)

MAP-NET™ CASE

The MAPNET2.1 Common Application Service Elements (CASE) is built on top of the MAPNET2.1 Session Service.

CASE is designed to support all the services provided by the lower ISO layers. In addition, MAP 2.1 CASE provides name-to-address translation for the user. By the use of the CASE service, a process can make a connection request to a remote process by using only the names of the processes. CASE takes these process names supplied by the user and resolves these names into network addresses and identification utilizing the services provided by the MAPNET2.1 Directory Service.

This greatly increases the ease-of-use of network Services provided by the underlying layers.

MAP-NET™ FILE TRANSFER, ACCESS AND MANAGEMENT (FTAM)

The FTAM Software in MAPNET2.1 provides remote file transfer capability. This capability is provided by the implementation of file request 'Initiator' module and a file request 'Responder' module. The Initiator intercepts file commands from the local user and transmits them across the LAN to the Responder at the node where the target file resides. The Responder receives, interprets, and executes the command acting as a user on its local node. File transfer between nodes is made possible by the implementation of a common set of file transfer protocols defined by the ISO FTAM Specification.

MAP-NET™ NETWORK MANAGEMENT FUNCTIONS (NMF)

The NMF meets or exceeds the MAP2.1 functionality for net management of each layer. The NMF interfaces to CASE, Session, Transport, Network, and Data Link layers. It provides three basic services:

— Read Net Management Object

— Set Net Management Object

— Event Notification

The NMF can be configured as a Net Manager for managing local or remote Net Agents or Net Agent for use by a remote Network Manager.

MAPNET2.1 FTAM allows a user to:

1) Create files on a remote node.
2) Write into files on a remote node.
3) Read files on a remote node.
4) Delete files on a remote node.
5) Get file attibutes on a remote node.

To perform the above functions the Initiator module should be configured in the user's node and the Responder should be configured in the remote target node. MAP-NET FTAM implementation allows a node to be 1) a file Initiator only, 2) a file Responder only and 3) both an Initiator and Responder.

MAP-NET™ and iNA 960 Software

MAPNET2.1 is designed to interface with iNA 960 Rel 2.0. iNA 960 Rel 2.0 provides the transport and network layers as required by the MAP specifications. Table 1 shows some examples of functions provided by MAPNET2.1 and iNA 960 Rel 2.0.

MAP2.1SXMSW—PROCONFIGURED LAYERS 3-7 MAP 2.1 SOFTWARE

MAP2.1SXMSW preconfigured 7 layer solution supports all seven layers on the iSBC 554 MULTIBUS® based commengine board. The services that are supplied by this preconfigured software package are FTAM, Directory Services, CASE, Session, Transport and Network layers. In order to provide maximum flexibility in interfacing user applications, the network management facility has been added. The preconfigured MAP software product is supplied with iRMX®86 device drivers, user interface utilities and the 7 layer conformance tested software.

OPERATING SYSTEM ENVIRONMENT

Figure 2 is a layout of the complete seven layer commengine. The preconfigured MAP software is downloaded on the iSBC 554 board. The user utilities can communicate with the seven layer commengine via the MULTIBUS Interface Protocol (MIP).

MIP is an Intel reliable process to process message delivery protocol between MULTIBUS processors. An implementation of the MIP protocol is provided on the iSBC 554 for communication with the host. The corresponding MIP/File Access Interface will have to be provided on the host side for communication with the iSBC 554. The user utilities include Directory Services, File Transfer and Net Management. The MIP/File Access Interface is available from Intel for the iRMX 86 Operating System and can be easily ported to other operating system environments.

Table 1. MAP2.1SXMSW and MAP-NET™/iNA 960 Rel 2.0 Services

Application	File Transfer, Access and Management (FTAM) provides remote operations on files (create, read, write, delete, get file attributes) Common Application Service Elements (CASE) supports all the services provided by the lower ISO layers provides name to address translation support Directory Services performs name to address conversion maintains local cache of resolved names two forms of Directory Service—client Service Agent for local data base and directory Service Agent for remote (master) data base
Presentation	Null
Session	Implements subset of ISO session 8327 specified by the MAP 2.1 specifications. Provides 'Graceful Close' 'graceful Close' allows the closing of a connection without any loss of queued requests it enhances the transport provided 'Close' which aborts a connection
Transport	Virtual circuit open: establish a virtual circuit database send connect: actively try to establish a virtual connection await connect: passively awaits the arrival of a connection request send: send a message receive: post a buffer to receive a message close: close a virtual circuit Datagram send: send a datagram message receive: post a buffer to receive a datagram message
Network	Internetworking routing between multiple lans segmentation/reassembly user defined routing tables Multiple subnets supported user supplied 802.3, 802.4
Data Link	Transmit: transmit a data link packet Receive: post a buffer to receive a data link packet Connect: make a data link logical connection (link service access point. IEEE802.4) Disconnect: disconnect a data link logical connection Change token bus address Add multicast address Delete multicast address Configure TBH
Network Management	Read/Clear/Set network objects (local/remote): read/clear/set local or remote MAP-NET/iNA 960 network parameters Read/Set network memory (local/remote): read/set memory of the local or a remote station useful in network debug process Boot consumer: requests a network boot server to load a boot file into this station Echo: Echo a packet between this station and another remote station on the network

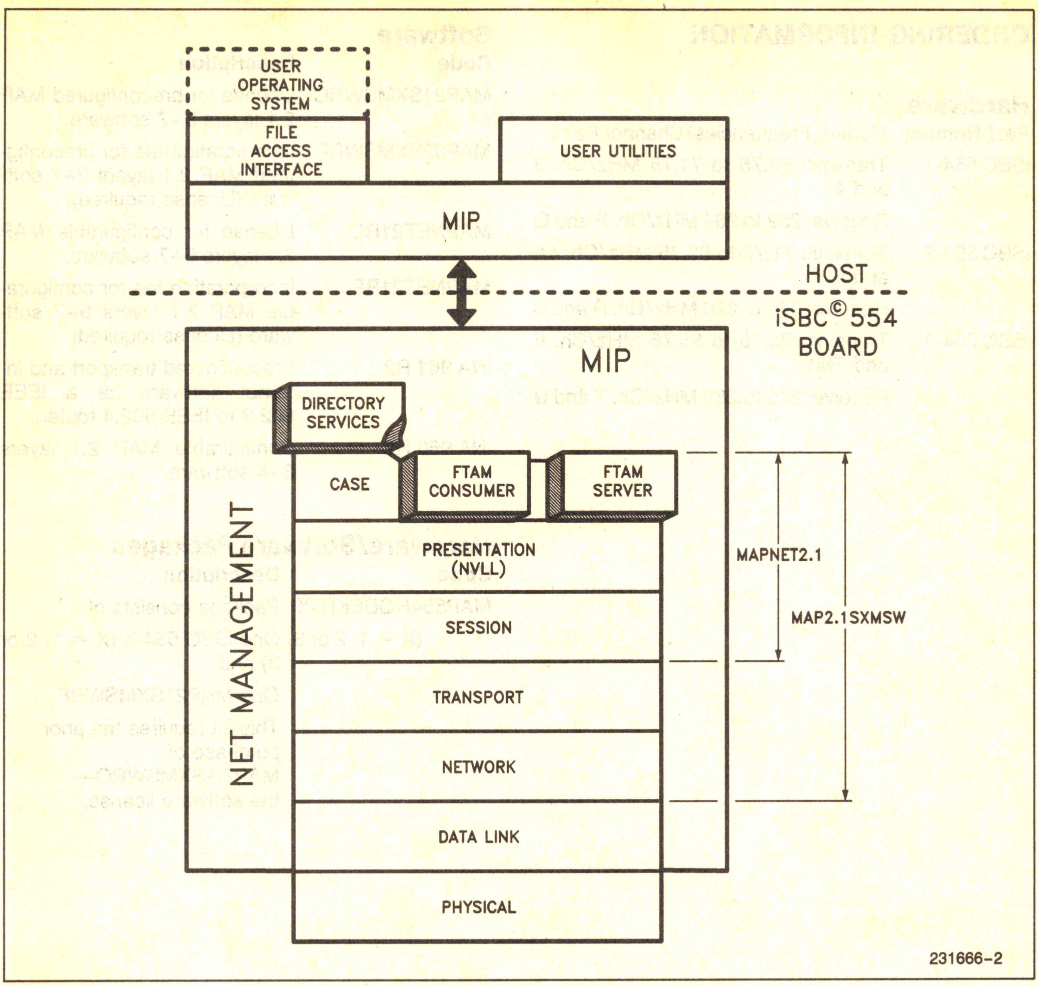


Figure 2. MAP-NET™/MAP2.1SXMSW User Interface

Available Literature

- iNA 960 Release 2.0 Programmers Reference Manual.
- iNA 960 Release 2.0 Configuration Guide
- iNA 960/MAP-NET Installation Guide
- iSBC 186/51 Hardware Reference Manual
- iSBC/iSXM 552 Hardware Reference Manual
- iSBC/iSXM 552A Hardware Reference Manual
- iSBC 554 Hardware Reference Manual
- MAP-NET Programmers Reference Manual
- RMX-NET Programmers Reference Manual
- iNA 960/961 Rel 2.0 Data Sheet
- iSBC 554 Data Sheet
- iSBC 552A Data Sheet

ORDERING INFORMATION

Hardware

Part Number	Modem Frequencies/Channel Pairs
iSBC 554-1	Transmit: 59.75 to 71.75 MHz/Ch. 3 and 4
	Receive: 252 to 264 MHz/Ch. P and Q
iSBC 554-2	Transmit: 71.75 to 83.75 MHz/Ch. 4A and 5
	Receive: 264 to 276 MHz/Ch. R and S
iSBC 554-3	Transmit: 83.75 to 95.75 MHz/Ch. 6 and FM1
	Receive: 276 to 288 MHz/Ch. T and U

Software

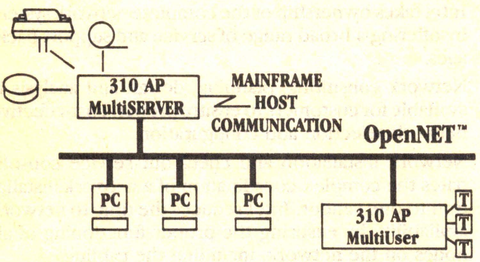
Code	Description
MAP21SXMSWRO	License for preconfigured MAP 2.1 layers 3–7 software.
MAP32SXMSWRF	Incorporation fee for preconfigured MAP 2.1 layers 3–7 software (License required).
MAPNET21RO	License for configurable MAP 2.1 layers 5–7 software.
MAPNET21RF	Incorporation fee for configurable MAP 2.1 layers 5–7 software (License required).
iNA 961 R2	Preconfigured transport and internet software for a IEEE 802.3 to IEEE 802.4 router.
iNA 960 R2	Configurable MAP 2.1 layers 3–4 software.

Hardware/Software Packages

Code	Description
MAP554NODEKIT-X (X = 1, 2 or 3)	Package consists of: One iSBC 554-X (X = 1, 2 or 3) and One MAP21SXMSWRF. This kit requires the prior purchase of MAP2.1SXMSWRO— the software license.

XENIX*-NET NETWORKING OpenNET™ PRODUCT FAMILY

- ▶ *Complete LAN Solution based entirely on standards*
- ▶ *Multiple operating system interoperation: XENIX, MS*-DOS, iRMX™, iNDX*
- ▶ *Existing applications distributed without change*
- ▶ *Comprehensive network services:*
  - — *Network File Access*
  - — *Remote Job Execution*
  - — *Network XENIX Mail*
  - — *Network Administration*
  - — *Virtual Terminal*
  - — *MS-DOS Virtual Terminal*
  - — *Print Spooling*



■ TOTAL LAN SOLUTION FOR MICROSYSTEM APPLICATIONS

OpenNET is Intel's Open System strategy and product family for local area networks (LANs). XENIX-NET represents the first truly integrated department service network to provide all the necessary hardware and software to link Intel microsystems, terminals, PCs, mainframes, minis, peripherals and software in one consistent, integrated system.

■ XENIX-NET NETWORK FILE ACCESS FOR TRANSPARENT INTEROPERATION

XENIX-NET provides transparent network file access (NFA) and additional network services to interoperate among various nodes on the LAN. XENIX-NET NFA runs under the XENIX 3.0 operating system from Intel. There are no special operating system calls to access remote files.

Applications and users make standard XENIX file access requests such as OPEN, CLOSE, READ and WRITE. XENIX-NET NFA transparently accesses files across the network. XENIX-NET NFA determines from the filename if the file is on a local storage device or remote across the network. Applications access remote files as if they were local; no modifications to applications software are required to run across the network.

XENIX-NET NFA makes networked microsystems look like one large integrated computer system with a single network-wide hierarchical file system.

■ XENIX-NET COMPLETE NETWORK SERVICES

In addition to transparent network file access, XENIX-NET makes available critical services to all nodes in a LAN providing for increased group productivity and system utilization.

Remote Job Execution. With Remote Job Execution, a user can execute a XENIX command stream at single or multiple remote nodes. Additionally, these command streams can be queued for execution at specific times throughout the day. This facility allows users to distribute and balance the workload logically throughout a network, completely utilizing the combined power of the network resources.

Network XENIX Mail. The XENIX Mail facility has been extended to transparently reach beyond a single XENIX system to remote nodes within a LAN. XENIX Mail users don't have to concern themselves with where a particular user resides on the network. Network XENIX Mail service provides the necessary routing and delivery throughout the network and through a UUCP link.

Virtual Terminal. Packaged as a separate network service, Virtual Terminal allows local XENIX users to "logon" to a remote Intel XENIX node within the network. This capability allows users to access all available resources and functions such as host communications and peripherals.

JUNE, 1986
ORDER NUMBER 270055-004

DOS-NET Virtual Terminal. Packaged as a separate network service, DOS-NET Virtual Terminal is an MS-DOS service which allows IBM PCs and compatibles connected through the OpenNET LAN to "logon" to any remote Intel XENIX system and access the multiuser applications and services (such as mail) available in that environment.

Easy Network Administration. XENIX-NET provides a complete set of interactive network configuration and maintenance utilities. With the addition of iBASE, Intel's menu driven business shell, network administration is further simplified by giving the network administrator a "window" to all nodes residing on a sub-network. A series of screens and menus prompts the administrator through network configuration and maintenance.

Print Spooling. XENIX-NET Print Spooling provides shared access to single or multiple printers distributed throughout a network. Expensive laser and letter-quality printers, for example, can be shared among numerous users from one site and need not be duplicated at each node in the network.

■ OpenNET LAN STANDARDS

Intel supports and drives LAN standards and technology for the microsystems and microcommunications industries. The OpenNET product family adheres to the International Standards Organization's (ISO) seven layer Open Systems Interconnect (OSI) model. Only complete products that conform to this model and are based on open and public standards carry the OpenNET name.

■ COMPLETE NETWORK SUPPORT AND SERVICE FROM A SINGLE SOURCE

Intel takes ownership of the complete network system by offering a broad range of service and support packages.

Network consulting, planning, design and analysis is available for customers to ensure proper, cost-effective network selection and configuration.

Network installation and check-out service consolidates the complex coordination of a network installation to one vendor. Intel reduces the time to network availability by ensuring the proper functioning of all nodes on the network, including the cabling.

Intel extended hardware/software service and support agreements are designed to support both Intel and non-Intel components of a network — making Intel the single point of contact for problems or questions relating to the network.

Finally, Intel offers complete training on XENIX-NET software, as well as for the entire OpenNET product line to make network users as productive as possible.

■ OpenNET XENIX-NET — THE TOTAL LAN SOLUTION

No other hardware and software LAN combination integrates such a breadth of services or offers a faster or more economical path to getting networked application systems that transform personal productivity into organizational efficiency.

ORDERING INFORMATION

Complete Network-Ready Systems

Complete XENIX-NET ready systems are available from Intel. SYS310-141 comes complete with an integrated LAN controller.SYS310-145 comes with an integrated LAN controller as well as an integrated mainframe Host Communication Controller for ASYNC and BYSNC communication protocols.

Any Series 300 microsystem from Intel may be upgraded to a network-ready system by adding an XNXNFAEKRIKIT option, or at initial order appending "XN" option designator for System 300 hardware configuration orders.

XENIX Networking Software and Kits	
XNXNFAEKRI	XENIX Networking and iNA 961 Object Software plus rights for 8 copies
XNXNFAEKRIKIT	XENIX Networking and iNA 961 Object Software plus an iSXM 552S Ethernet controller for pass-through product
DOSNETVTSKRI	PC terminal emulator that enables a PC user to "login" directly to a XENIX system running XENIX virtual terminal
XNXNETVTSKRI	Provides XENIX-to-XENIX virtual terminal capabilities
LAN Hardware	
iSXM552	Ethernet COMMengine plus one iNA 961 Software Incorporation Fee
iMDX457	Ethernet Transceiver Cable
iMDX3015	Ethernet Transceiver
iMDX3016-1	Ethernet Cable
iDCM911-1	Intellink
PCLINK	Includes the Network Interface Unit (NIU) add-in IEEE 802.3 "Ethernet" controller for IBM PC and compatibles, preconfigured iNA 961 ISO transport software for NIU and MS NET network file access software for PC DOS/MS-DOS PC or compatible.

Serial Communication Boards and Software

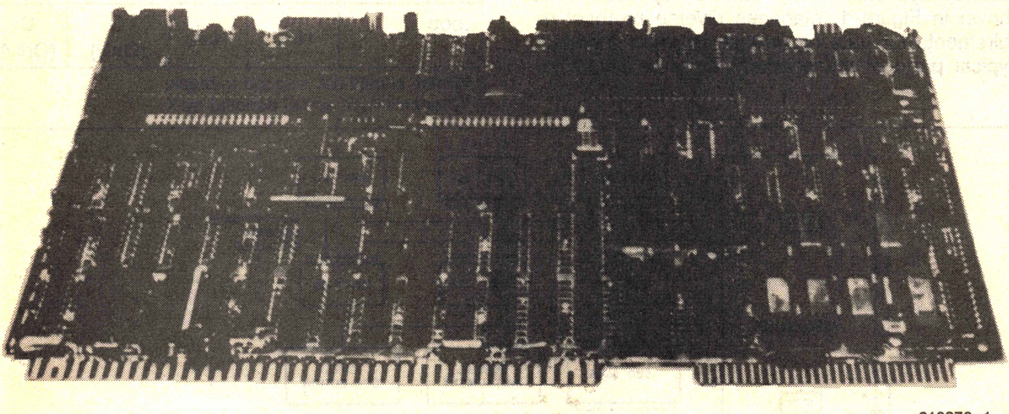
8

intel®

iSBC® 88/45 ADVANCED DATA COMMUNICATIONS PROCESSOR BOARD

- Three HDLC/SDLC Half/Full-Duplex Communication Channels—Optional ASYNC/SYNC on Two Channels
- Supports RS232C (Including Modem Support), CCITT V.24, or RS422A/449 Interfaces
- On-Board DMA Supports 800K Baud Operation
- Self-Clocking NRZI SDLC Loop Data Link Interface
  - Point-to-Point
  - Multidrop
- Software Programmable Baud Rate Generation
- 8088 (8088-2) Microprocessor Operates at 8 MHz
- iSBC® 337 Numeric Data Processor Option Supported
- 16K Bytes Static RAM (12K Bytes Dual-Ported)
- Four 28-Pin JEDEC Sites for EPROM/RAM Expansion; Four Additional 28-Pin JEDEC Sites Added with iSBC® 341 Board
- Two iSBX™ Bus Connectors
- MULTIBUS® Interface Supports Multimaster Configuration

The iSBC 88/45 Advanced Data Communications Processor (ADCP) Board adds 8 MHz, 8088 (8088-2) 8-bit microprocessor-based communications flexibility to the Intel line of OEM microcomputer systems. The iSBC 88/45 ADCP board offers asynchronous, synchronous, SDLC, and HDLC serial interfaces for gateway networking or general purpose solutions. The iSBC 88/45 ADCP board provides the CPU, system clock, EPROM/RAM, serial I/O ports, priority interrupt logic, and programmable timers to facilitate higher-level application solutions.



210372-1

November 1986
Order Number: 210372-002

FUNCTIONAL DESCRIPTION

Three Communication Channels

Three programmable HDLC/SDLC serial interfaces are provided on the iSBC 88/45 ADCP board. The SDLC interface is familiar to IBM system and terminal equipment users. The HDLC interface is known by users of CCITT's X.25 packet switching interface.

One channel utilizes an Intel 8273 controller to manage the serial data transfers. Accepting the 8-bit data bytes from the local bus, the 8273 controller translates the data into the HDLC/SDLC format. The channel operates in half/full-duplex mode.

In addition to the synchronous mode, the 8273 controller operates asynchronously with NRZI encoded data which is found in systems such as the IBM 3650 Retail Store System. An SDLC loop configuration using iSBX 352 and iSBC 88/45 products is shown in Figure 1.

The two additional channels utilize the Intel 8274 Multi-Protocol Serial Controller (MPSC). The MPSC provides two independent half/full-duplex serial channels which provide asynchronous, synchronous, HDLC or SDLC protocol operations. The sync and async protocol operations are commonly used to communicate with inexpensive terminals and systems.

The three serial channels of the iSBC 88/45 ADCP board offer communications capability to manage a gateway application. The gateway application, as shown in Figure 1, manages diverse protocol requirements for data movement between channels. Typical protocol management software layers implemented by the user include SNA terminal interfaces to IBM systems.

On-Board DMA

For high-speed communications, one MPSC channel has a DMA capacity to support an 800K baud rate. The second channel attached to the MPSC is capable of simultaneous 800K baud operation when configured with DMA capability, but is connected to an RS232C interface which is defined as 20K baud maximum. Figure 2 shows an RS422A/449 multidrop application which supports high-speed operation.

Interfaces Supported

The iSBC 88/45 ADCP board provides an excellent foundation to support these electrical and diverse software drivers protocol interfaces. The control lines, serial data lines, and signal ground lines are brought out to the three double-edge connectors. Figure 3 shows the cable to connector construction. Two connectors are pre-configured for RS422A/449. All three channels are configurable for RS232C/CCITT V.24 interfaces as shown in Table 1.

Table 1. iSBC® 88/45 Supported Configurations

Connection	Synchronous		Asynchronous	
	Modem	Direct	Modem*	Direct
Point-to-Point	X**	X	X	X
Multidrop	X	X	X	X
Loop	N.A.	N.A.	C (Only)	C (Only)

*Modem should not respond to break.
**Channels A, B, and C denoted by X.

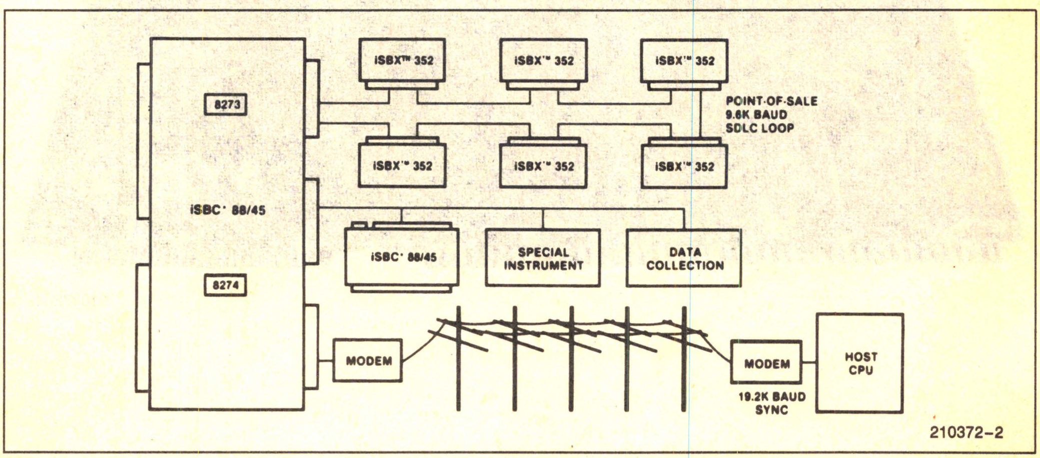
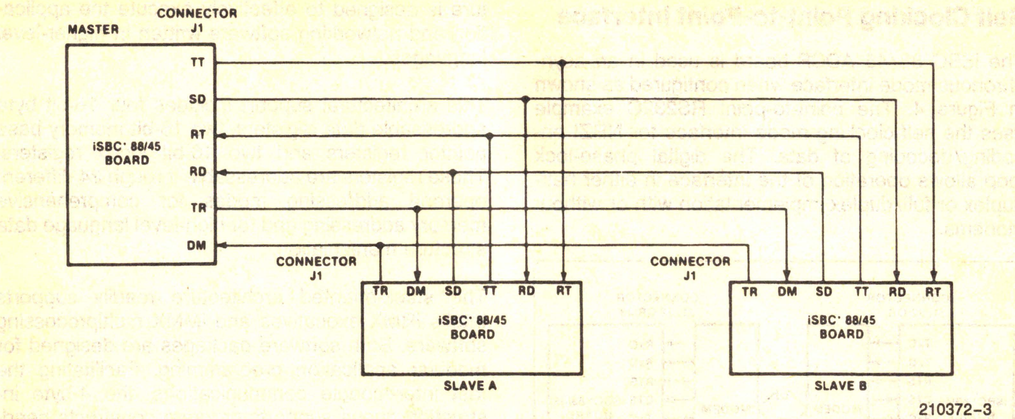


Figure 1. iSBC® 88/45 Gateway Processor Example



NOTE:
The last slave device in the system must contain termination resistors on all signal lines received by the slave board. The master device contains bias resistors on all signal lines.

Figure 2. Synchronous Multidrop Network Configuration Example—RS422A

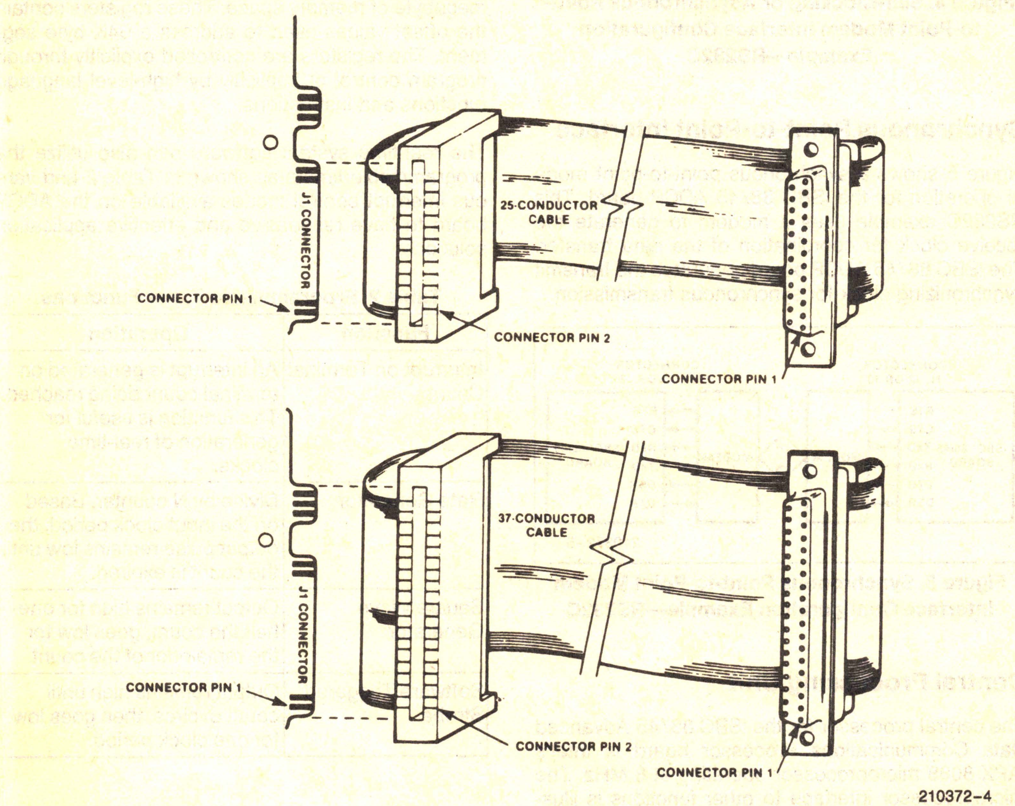


Figure 3. Cable Construction and Installation for RS232C and RS422A/449 Interface

Self Clocking Point-to-Point Interface

The iSBC 88/45 ADCP board is used in an asynchronous mode interface when configured as shown in Figure 4. The point-to-point RS232C example uses the self-clocking mode interface for NRZI encoding/decoding of data. The digital phase-lock loop allows operation of the interface in either half-duplex or full/duplex implementation with or without modems.

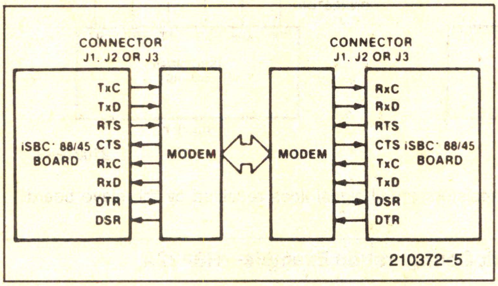


Figure 4. Self-Clocking or Asynchronous Point-to-Point Modem Interface Configuration Example—RS232C

Synchronous Point-to-Point Interface

Figure 5 shows a synchronous point-to-point mode of operation for the iSBC 88/45 ADCP board. This RS232C example uses a modem to generate the receive clock for coordination of the data transfer. The iSBC 88/45 ADCP board generates the transmit synchronizing clock for synchronous transmission.

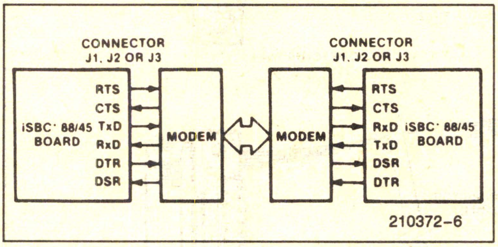


Figure 5. Synchronous Point-to-Point Modem Interface Configuration Example—RS232C

Central Processing Unit

The central processor for the iSBC 88/45 Advanced Data Communications Processor board is Intel's iAPX 8088 microprocessor operating at 8 MHz. The microprocessor interface to other functions is illustrated in Figure 6. The microprocessor architecture is designed to effectively execute the application and networking software written in higher-level languages.

This architectural support includes four 16-bit byte addressable data registers, two 16-bit memory base pointer registers and two 16-bit index registers. These registers are addressable through 24 different operand addressing modes for comprehensive memory addressing and for high-level language data structure manipulation.

The stack-oriented architecture readily supports Intel's iRMX executives and iMMX multiprocessing software. Both software packages are designed for modular application programming. Facilitating the fast inter-module communications, the 4-byte instruction queue supports program constructs needed for real-time systems.

Since programs are segmented between pure procedure and data, four segment registers (code, stack, data, extra) are available for addressing 1 megabyte of memory space. These registers contain the offset values used to address a 64K byte segment. The registers are controlled explicitly through program control or implicitly by high-level language functions and instructions.

The real-time system software can also utilize the programmable timers as shown in Table 2 and various interrupt control modes available on the ADCP board to have responsive and effective application solutions.

Table 2. Programmable Timer Functions

Function	Operation
Interrupt on Terminal Count	An interrupt is generated on terminal count being reached. This function is useful for generation of real-time clocks.
Rate Generator	Divide by N counter. Based on the input clock period, the output pulse remains low until the count is expired.
Square Wave Generator	Output remains high for one-half the count, goes low for the remainder of the count.
Software Triggered Strobe	Output remains high until count expires, then goes low for one clock period.

Numeric Data Processor Extension

The 8088 instruction set includes 8-bit and 16-bit signed and unsigned arithmetic operators for bi-

nary, BCD, and unpacked ASCII data. For enhanced numerics processing capability, the iSBC 337 MULTIMODULE Numeric Data Processor extends the 8088 architecture and data set(1).

The extended numerics capability includes over 60 numeric instructions offering arithmetic, trigonometric, transcendental, logarithmic, and exponential instructions. Many math-oriented applications utilize the 16-, 32-, and 64-bit integer, 32- and 64-bit floating point, 18-digit packed BCD, and 80-bit temporary data types.

16K Bytes Static Ram

The iSBC 88/45 ADCP board contains 16K bytes of high-speed static RAM, with 12K bytes dual-ported which is addressable from other MULTIBUS devices. When coupled with the high-speed DMA capability of the iSBC 88/45 ADCP board, the dual-ported memory provides effective data communication buffers. The dual-ported memory is useful for interprocessor message transfers.

NOTE:
1. The iSBC 337 board requires the iSBC 88/45 ADCP board can be jumpered to provide 4 MHz operation.

Interrupt Capability

The iSBC 88/45 ADCP board provides nine vectored interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line. The additional eight interrupt levels are vectored via the Intel 8259A Programmable Interrupt Controller (PIC). As shown in Table 3, four priority processing modes are available to match interrupt servicing requirements. These modes and priority assignments are dynamically configurable by the system software.

Table 3. Programmable Interrupt Modes

Mode	Operation
Nested	Interrupt request line priorities fixed; interrupt 0 is the highest and 7 is the lowest.
Auto-Rotating	The interrupt priority rotates; once an interrupt is serviced it becomes the lowest priority.
Specific Priority	System software assigns lowest level priority. The other levels are sequenced based on the level assigned.
Polled	System software examines priority interrupt via interrupt status register.

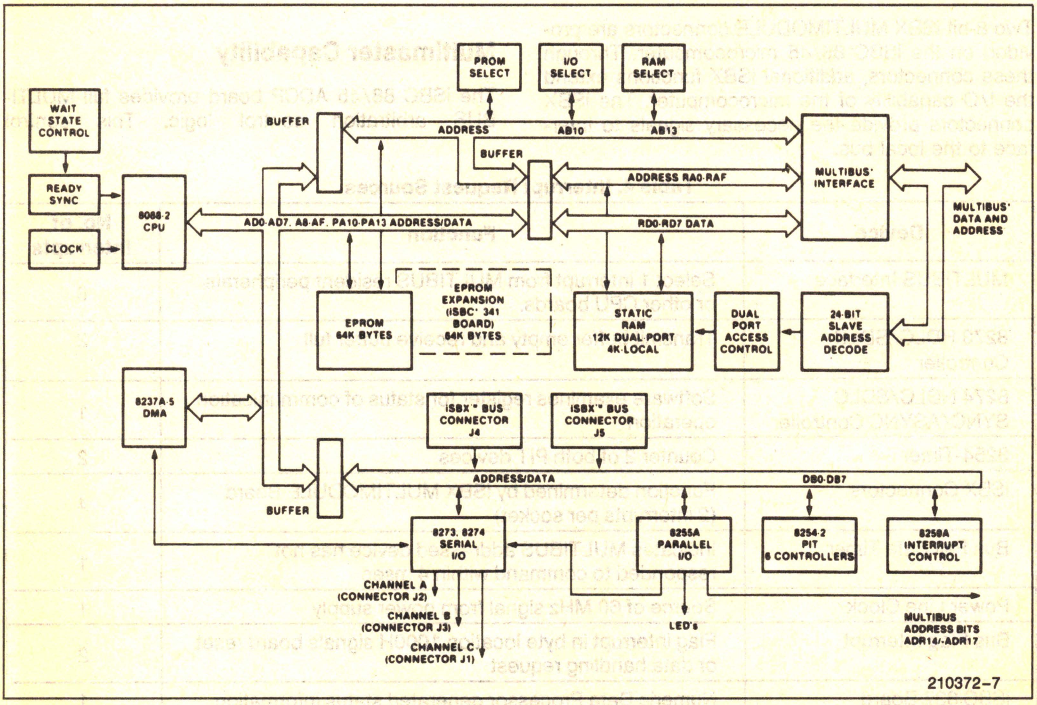


Figure 6. Block Diagram of the ISBC® 88/45 ADCP Board

Interrupt Request Generation

Listed in Table 4 are the devices and functions supported by interrupts on the iSBC 88/45 ADCP board. All interrupt signals are brought to the interrupt jumper matrix. Any of the 23 interrupt sources are strapped to the appropriate 8259A PIC request level. The PIC resolves requests according to the software selected mode and, if the interrupt is unmasked, issues an interrupt to the CPU.

EPROM/RAM Expansion

In addition to the on-board RAM, the iSBC 88/45 ADCP board provides four 28-pin JEDEC sockets for EPROM expansion. By using 2764 EPROMs, the board has 32K bytes of program storage. Three of the JEDEC standard sockets also support byte-wide static RAMs or iRAMs; using 8K x 8 static RAMs provides an additional 24K bytes of RAM.

Inserting the optional iSBC 341 MULTIMODULE EPROM expansion board onto the iSBC 88/45 ADCP board provides four additional 28-pin JEDEC sites. This expansion doubles the available program storage or extends the RAM capability by 32K bytes.

iSBX™ MULTIMODULE™ Expansion

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 88/45 microcomputer. Through these connectors, additional iSBX functions extend the I/O capability of the microcomputer. The iSBX connectors provide the necessary signals to interface to the local bus.

In addition to specialized or custom designed iSBX boards, the customer has a broad range of Intel iSBC MULTIMODULEs available, including parallel I/O, analog I/O, iEEE 488 GPIB, floppy disk, magnetic bubbles, video, and serial I/O boards.

The serial I/O MULTIMODULE boards include the iSBX 351 (one ASYNC/SYNC serial channel) the iSBX 352 (one HDLC/SDLC serial channel) and the iSBX 354 (two SYNC/ASYNC, HDLC/SDLC serial channels) boards. Adding two iSBX 352 MULTIMODULE boards to the iSBC 88/45 ADCP provides a total of five HDLC/SDLC channels.

MULTIBUS® Multimaster Capabilities

OVERVIEW

The MULTIBUS system is Intel's industry standard microcomputer bus structure. Both 8- and 16-bit single board computers are supported on the MULTIBUS structure with 24 address and 16 data lines. In addition to expanding functions contained on a single board computer (e.g., memory and digital I/O), the MULTIBUS structure allows very powerful distributed processing configurations with multiple processors, intelligent slaves, and peripheral boards.

Multimaster Capability

The iSBC 88/45 ADCP board provides full MULTIBUS arbitration control logic. This control

Table 4. Interrupt Request Sources

Device	Function	No. of Interrupts
MULTIBUS Interface	Select 1 interrupt from MULTIBUS resident peripherals or other CPU boards.	8
8273 HDLC/SDLC Controller	Transmit buffer empty and receive buffer full	2
8274 HDLC/SDLC SYNC/ASYNC Controller	Software examines register for status of communication operation	1
8254-Timer	Counter 2 of both PIT devices	2
iSBX Connectors	Function determined by iSBX MULTIMODULE Board (2 interrupts per socket)	4
Bus Fail Safe Timer	Indicates MULTIBUS addressed device has not responded to command within 4 msec	1
Power Line Clock	Source of 60 MHz signal from power supply	1
Bus Flag Interrupt	Flag interrupt in byte location 1000H signals board reset or data handling request	2
iSBC 337 Board	Numeric Data Processor generated status information	1
8237A-5	Signals end of 8237 DMA operation	1

logic allows up to three iSBC 88/45 ADCP boards or other bus masters, including iSBC 286, iSBC 86 and iSBC 86 family boards to share the system bus using a serial (daisy chain) priority scheme. By using an external parallel priority decoder, the MULTIBUS system bus could be shared among sixteen masters.

The Intel standard MULTIBUS Interprocessor Protocol (MIP) software, implemented as the Intel iMMX 800 package for iRMX 86 and iRMX 88 Real-Time Executives, fully supports multiple 8- and 16-bit distributed processor functions. The software manages the message passing protocol between microprocessors.

System Development Capabilities

The application development cycle for an iSBC 88/45 ADCP board is reduced and simplified through the usage of several Intel tools. The tools include the Intellec Series Microcomputer Development System, the ICE-88 In-Circuit Emulator, the iSDM 86 debug monitor software, and the iRMX 86 and iRMX 88 run-time support packages.

The Intellec Series Microcomputer Development System offers a complete development environment for the iSBC 88/45 software. In addition to the operating system, assembler, utilities and application debugger features provided with the system, the user optionally can utilize higher-level languages like PL/M, PASCAL, and FORTRAN.

The ICE-88 In-Circuit Emulator provides a link between the Intellec system and the target iSBC 88/45-based system for code loading and execution. The ICE-88 package assists the developer with the debugging and system integrating processes.

Run-Time Building Blocks

Intel offers run-time foundation software to support applications which range from general purpose to high-performance solutions. The iRMX 88 Real-time Multitasking Executive provides a multitasking structure which includes task scheduling, task management, intertask communications, and interrupt servicing for high-performance applications. The highly configurable modules make the system tailoring job easier whether one uses the compact executive or the complete executive with its variety of peripheral devices supported.

The iRMX 86 Operating System provides a very rich set of features and options to support sophisticated applications solutions. In addition to supporting real-time requirements, the iRMX 86 Operating System has a powerful, but easy-to-use human interface. When added to the sophisticated I/O system, the iRMX 86 Operating System is readily extended to support assembler, PL/M, PASCAL, and FORTRAN software development environments. The modular building block software lends itself well to customized application solutions.

SPECIFICATIONS

Word Size

Instruction: 8, 16, 24, or 32 bits
Data: 8 or 16 bits

System Clock

8 MHz: ±0.1%

NOTE:
Jumper selectable for 4 MHz operation with iSBC 337 Numeric Data Processor module or ICE-88 product.

Cycle Time

Basic Instruction Cycle at 8.00 MHz: 1.25 μs, 250 ns (assumes instruction in the queue)

NOTE:
Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles).

Memory Cycle Time

RAM: 500 ns (no wait states)
EPROM: jumper selectable from 500 ns to 625 ns.

On-Board RAM*

K Bytes	Hex Address Range
16 (total)	0000–3FFF
12 (dual-ported)	1000–3FFF

*Four iSBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (3 sockets); iSBC 341 (4 sockets)

Environmental Characteristics

Temperature: 0°C to +55°C, free moving air across the base board and MULTIMODULE board

Humidity: 90%, non-condensing

Physical Characteristics

Width: 30.48 cm (12.00 in)
Length: 17.15 cm (6.75 in)
Height: 1.50 cm (0.59 in)
Weight: 6.20 gm (22 oz)

Memory Capacity/Addressing

On-Board EPROM*

Device	Total K Bytes	Hex Address Range
2716	8	FE000–FFFFF
2732A	16	FC000–FFFFF
2764	32	F8000–FFFFF
27128	64	F0000–FFFFF

With optional iSBC® 341 MULTIMODULE™ EPROM

Device	Total K Bytes	Hex Address Range
2716	16	FC000–FFFFF
2732A	32	F8000–FFFFF
2764	64	F0000–FFFFF
27128	128	E0000–FFFFF

*Four iSBC 88/45 EPROM sockets support JEDEC 24/28-pin standard EPROMs and RAMs (static and iRAM, 3 sockets); iSBC 341 sockets also support EPROMs and RAMs.

Timer Input Frequency—8.00 MHz ±0.1%

Interfaces

iSBX™ Bus—All signals TTL compatible

Serial RS232C Signals—

CTS	CLEAR TO SEND
DSR	DATA SET READY
DTE TXC	TRANSMIT CLOCK
DTR	DATA TERMINAL READY
FG	FRAME GROUND
RTS	REQUEST TO SEND
RXC	RECEIVE CLOCK
RXD	RECEIVE DATA
SG	SIGNAL GROUND
TXD	TRANSMIT DATA

Serial RS422A/449 Signals—

CS	CLEAR TO SEND
DM	DATA MODE
RC	RECEIVE COMMON
RD	RECEIVE DATA
RS	REQUEST TO SEND
RT	RECEIVE TIMING
SC	SEND COMMON
SD	SEND DATA
SG	SIGNAL GROUND
TR	TERMINAL READY
TT	TERMINAL TIMING

Electrical Characteristics

DC Power Dissipation—28.3 Watts

DC Power Requirements

Configuration	Current Requirements (All Voltages ±5%) +5V	+12V	−12V
Without EPROM(1)	5.1A	20 mA	20 mA
With 8K EPROM (Using 2716)	+0.14A	—	—
With 16K EPROM (Using 2732A)	+0.20A	—	—
With 32K EPROM (Using 2764)	+0.24A	—	—
With 64K EPROM (Using 27128)	+0.24A	—	—

NOTE:
1. AS SHIPPED—no EPROMs in sockets, no iSBC 341 module. Configuration includes terminators for two RS422A/449 and one RS232C channels.

Serial Communication Characteristics

Channel	Device	Supported Interface	Max. Baud Rate
A	8274(1)	RS442A/449 RS232C CCITT V.24	800K SDLC/HDLC 125K Synchronous 50K Asynchronous
B	8274	RS232C CCITT V.24	125K Synchronous(2) 50K Asynchronous
C	8273(3)	RS442A/449 RS232C CCITT V.24	64K SDLC/HDLC(3) 9.6K SELF CLOCKING

NOTES:
1. 8274 supports HDLC/SDLC/SYNC/ASYNC multiprotocol
2. Exceed RS232C/CCITT V.24 rating of 20K baud
3. 8273 supports HDLC/SDLC

BAUD RATE EXAMPLES (Hz)

8254 Timer Divide Count N	Synchronous K Baud	Asynchronous K Baud ÷16	÷32	÷64
10	800	50.0	25.0	12.5
26	300	19.2	9.6	4.8
31	256	16.1	8.06	4.03
52	154	9.6	4.8	2.4
104	76.8	4.8	2.4	1.2
125	64	4.0	2.0	1.0
143	56	3.5	1.7	0.87
167	48	3.0	1.5	0.75
417	19.2	—	—	—
833	9.6	—	—	—
EQUATION	$\frac{8,000,000}{N}$	$\frac{500K}{N}$	$\frac{250K}{N}$	$\frac{125K}{N}$

SERIAL INTERFACE CONNECTORS

Interface	Mode(1)	MULTIMODULE™ Edge Connector	Cable	Connector
RS232C	DTE	26-pin(4), 3M-3462-0001	3M(2)-3349/25	25-pin(6), 3M-3482-1000
RS232C	DCE	26-pin(4), 3M-3462-0001	3M(2)-3349/25	25-pin(6), 3M-3483-1000
RS449	DTE	40-pin(5), 3M-3464-0001	3M(3)-3349/37	37-pin(7), 3M-3502-1000
RS449	DCE	40-pin(5), 3M-3464-0001	3M(3)-3349/37	37-pin(7), 3M-3503-1000

NOTES:

1. DTE—Data Terminal Equipment Mode (male connector); DCE—Data Circuit Equipment mode (female connector) requires line swaps.
2. Cable is tapered at one end to fit the 3M-3462 connector.
3. Cable is tapered to fit 3M-3464 connector.
4. Pin 26 of the edge connector is not connected to the flat cable.
5. Pins 38, 39, and 40 of the edge connector are not connected to the flat cable.
6. May be used with the cable housing 3M-3485-1000.
7. Cable housing 3M-3485-4000 may be used wih the connector.

Line Drivers (Supplied)

Device	Characteristic	Qty	Installed
1488	RS232C	3	1
1489	RS232C	3	1
3486	RS422A	2	2
3487	RS422A	2	2

Reference Manual

143824—iSBC 88/45 Advanced Data Communications Processor Board Hardware Reference Manual (not supplied).

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

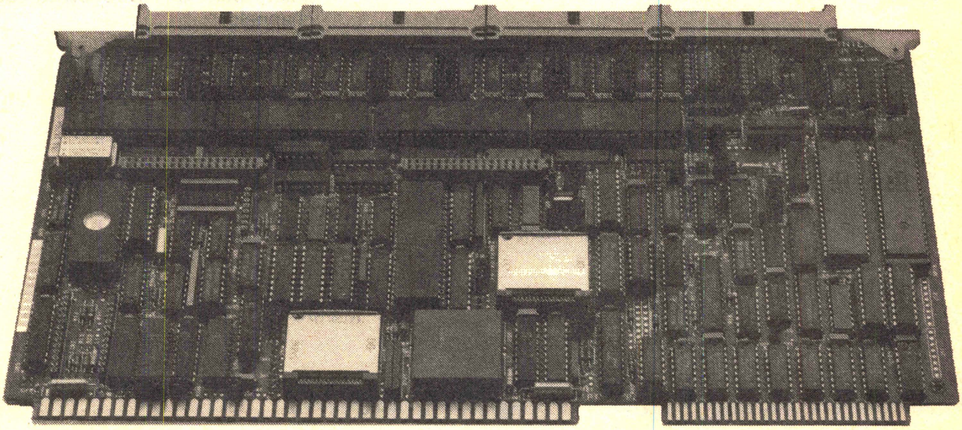
ORDERING INFORMATION

Part Number	Description
SBC 88/45	8-bit 8088-based Single Board Computer with 3 HDLC/SDLC serial channels

iSBC® 188/56 ADVANCED COMMUNICATING COMPUTER

- iSBC® Single Board Computer or Intelligent Slave Communication Board
- 8 Serial Communications Channels, Expandable to 12 Channels on a Single MULTIBUS® Board
- 8 MHz 80188 Microprocessor
- Supports RS232C Interface on 6 Channels, RS422A/449 or RS232C Interface Configurable on 2 Channels
- Supports Async, Bisync HDLC/SDLC, On-Chip Baud Rate Generation, Half/Full-Duplex, NRZ, NRZI or FM Encoding/Decoding
- 7 On-Board DMA Channels for Serial I/O, 2 80188 DMA Channels for the iSBX™ MULTIMODULE™ Board
- MULTIBUS Interface for System Expansion and Multimaster Configuration
- Two iSBX Connectors for Low Cost I/O Expansion
- 256K Bytes Dual-Ported RAM On-Board
- Two 28-pin JEDEC PROM Sites Expandable to 6 Sites with the iSBC 341 MULTIMODULE Board for a Maximum of 192K Bytes EPROM
- Resident Firmware to Handle up to 12 RS232C Async Lines

The iSBC 188/56 Advanced Communicating Computer (COMMputer™) is an intelligent 8-channel single board computer. This iSBC board adds the 8 MHz 80188 microprocessor-based communications flexibility to the Intel line of OEM microcomputer systems. Acting as a stand-alone CPU or intelligent slave for communication expansion, this board provides a high performance, low-cost solution for multi-user systems. The features of the iSBC 188/56 board are uniquely suited to manage higher-layer protocol requirements needed in today's data communications applications. This single board computer takes full advantage of Intel's VLSI technology to provide state-of-the-art, economic, computer based solutions for OEM communications-oriented applications.



280715-1

*IBM is a registered trademark of International Business Machines
*UNIX is a trademark of Bell Laboratories
*XENIX is a trademark of Microsoft Corporation

November 1986
Order Number: 280715-002

OPERATING ENVIRONMENT

The iSBC 188/56 COMMputer™ features have been designed to meet the needs of numerous communications applications. Typical applications include:

1. Terminal/cluster controller
2. Front-end processor
3. Stand-alone communicating computer

Terminal/Cluster Controller

A terminal/cluster controller concentrates communications in a central area of a system. Efficient handling of messages coming in or going out of the system requires sufficient buffer space to store messages and high speed I/O channels to transmit messages. More sophisticated applications, such as cluster controllers, also require character and format conversion capabilities to allow different types of terminals to be attached.

The iSBC 188/56 Advanced Communicating Computer is well suited for multi-terminal systems (see Figure 1). Up to 12 serial channels can be serviced in multi-user or cluster applications by adding two iSBX 354 MULTIMODULE boards. The dual-port RAM provides a large on-board buffer to handle incoming and outgoing messages at data rates up to 19.2K baud. Two channels are supported for continuous data rates greater than 19.2K baud. Each serial channel can be individually programmed for different baud rates to allow system configurations with differing terminal types. The firmware supplied on the iSBC 188/56 board supports up to 12 asynchronous RS232C serial channels, provides modem control and performs power-up diagnostics. The high performance of the on-board CPU provides intelligence to handle protocols and character handling typically assigned to the system CPU. The distribution of intelligence results in optimizing system performance by releasing the system CPU of routine tasks.

Front-End Processor

A front-end processor off-loads a system's central processor of tasks such as data manipulation and text editing of characters collected from the attached terminals. A variety of terminals require flexible terminal interfaces. Program code is often dynamically downloaded to the front-end processor from the system CPU. Downloading code requires sufficient memory space for protocol handling and program code. Flow control and efficient handling of interrupts require an efficient operating system to manage the hardware and software resources.

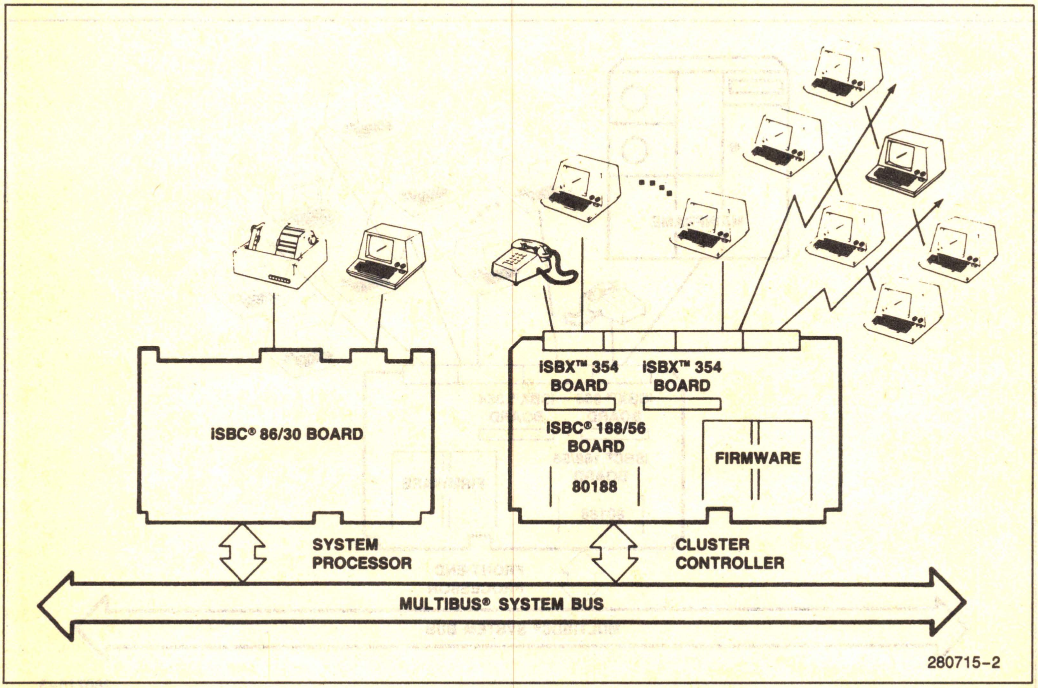


Figure 1. Terminal/Cluster Controller Application

The iSBC 188/56 board features are designed to provide a high performance solution for front-end processor applications (see Figure 2). A large amount of random access memory is provided for dynamic storage of program code. In addition, local memory sites are available for storing routine programs such as X.25, SNA or bisync protocol software. The serial channels can be configured for links to mainframe systems, point-to-point terminals, modems or multidrop configurations.

Stand-Alone COMMputer™ Application

A stand-alone communication computer is a complete computer system. The CPU is capable of managing the resources required to meet the needs of multi-terminal, multi-protocol applications. These applications typically require multi-terminal support, floppy disk control, local memory allocation, and program execution and storage.

To support stand-alone applications, the iSBC 188/56 COMMputer board uses the computational capabilities of an on-board CPU to provide a high-speed system solution controlling 8 to 12 channels of serial I/O (see Figure 3). The local memory available is large enough to handle special purpose code, execution code and routine protocol software.

The MULTIBUS interface can be used to access additional system functions. Floppy disk control and graphics capability can be added to the iSBC stand-alone computer through the iSBX connectors.

ARCHITECTURE

The four major functional areas are Serial I/O, CPU, Memory and DMA. These areas are illustrated in Figure 4.

Serial I/O

Eight HDLC/SDLC serial interfaces are provided on the iSBC 188/56 board. The serial interface can be expanded to 12 channels by adding 2 iSBX 354 MULTIMODULE boards. The HDLC/SDLC interface is compatible with IBM* system and terminal equipment and with CCITT's X.25 packet switching interface.

Four 82530 Serial Communications Controllers (SCC) provide eight channels of half/full duplex serial I/O. Six channels support RS232C interfaces. Two channels are RS232C/422/449 configurable and can be tri-stated to allow multidrop networks. The 82530 component is designed to satisfy several serial communications requirements; asynchronous,

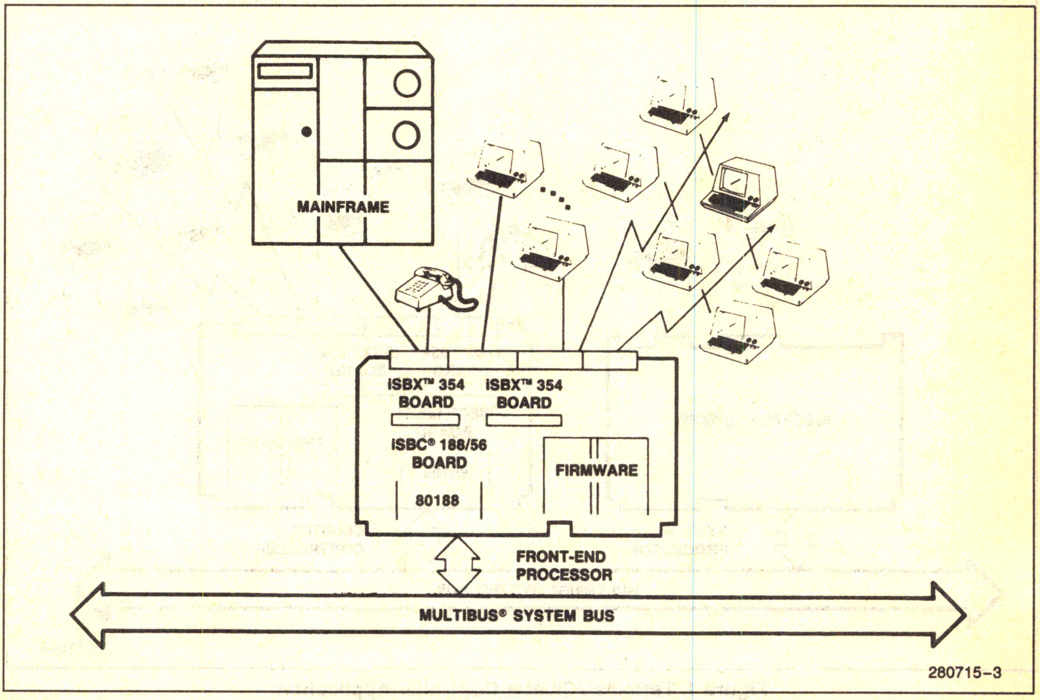


Figure 2. Front-End Processor Application

byte-oriented synchronous (HDLC/SDLC) modes. The increased capability at the serial controller point results in off-loading the CPU of tasks formerly assigned to the CPU or its associated hardware. Configurability of the 82530 allows the user to configure it to handle all asynchronous data formats regardless of data size, number of start or stop bits, or parity requirements. An on-chip baud rate generator allows independent baud rates on each channel.

The clock can be generated either internally with the SCC chip, with an external clock or via the NRZ1 clock encoding mechanism.

All eight channels can be configured as Data Terminal Equipment (DTE) or Data Communications Equipment (DCE). Table 1 lists the interfaces supported.

Table 1. iSBC® 188/56 Interface Support

Connection	Synchronous	Asynchronous
	Modem to Direct	Modem to Direct
Point-to-Point	X** Channels	X Channels
Multidrop	0 and 1	0 and 1
Loop	X	N/A

**All 8 channels are denoted by X.

Central CPU

The 80188 central processor component provides high performance, flexibility and powerful processing. The 80188 component is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The 80188 is upward compatible with 86 and 186 software.

The 80188/82530 combination with on-board PROM/EPROM sites, and dual-port RAM provide the intelligence and speed to manage multi-user, multi-protocol communication operations.

Memory

There are two areas of memory on-board: dual-port RAM and universal site memory. The iSBC 188/56 board contains 256K bytes of dual-port RAM that is addressable by the 80188 on-board. The dual-port memory is configurable anywhere in a 16M byte address space on 64K byte boundaries as addressed from the MULTIBUS port. Not all of the 256K bytes are visible from the MULTIBUS bus side. The amount of dual-port memory visible to the

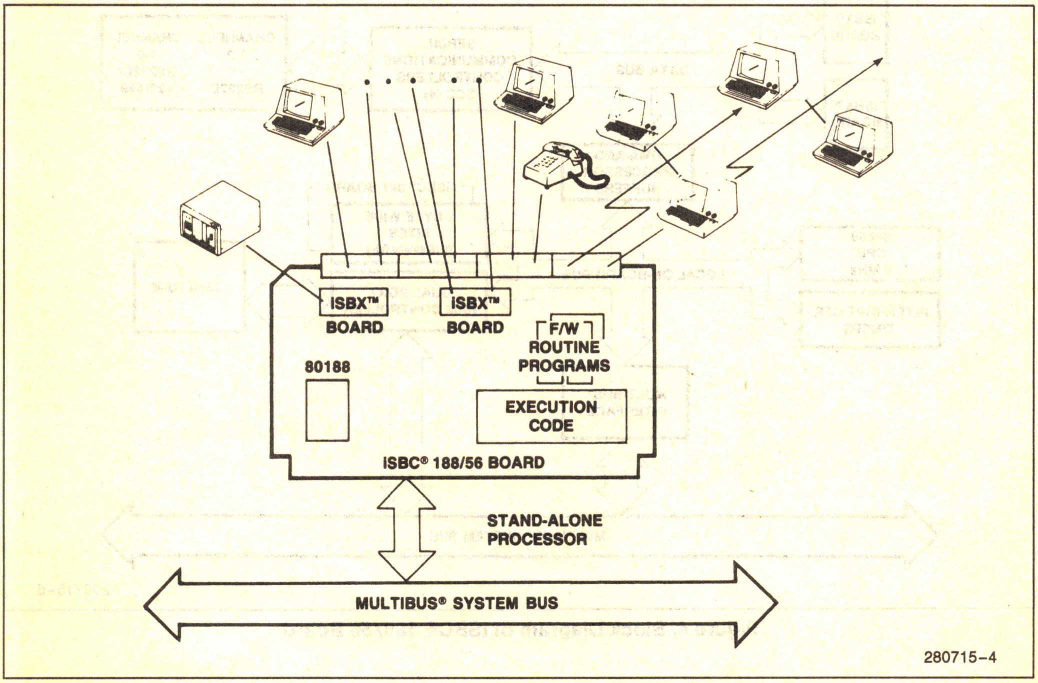


Figure 3. Stand-Alone COMMputer™ Application

MULTIBUS side can be set (with jumpers) to none, 16K bytes, or 48K bytes. In a multiprocessor system these features provide local memory for each processor and shared system memory configurations where the total system memory size can exceed one megabyte without addressing conflicts.

The second area of memory is universal site memory providing flexible memory expansion. Two 28-pin JEDEC sockets are provided. One of these sockets is used for the resident firmware as described in the FIRMWARE section.

The default configuration of the boards supports 16K byte EPROM devices such as the Intel 27128 component. However, these sockets can contain ROM, EPROM, Static RAM, or EEPROM. Both sockets must contain the same type of component (i.e. as the first socket contains an EPROM for the resident firmware, the second must also contain an EPROM with the same pinout). Up to 32K bytes can be addressed per socket giving a maximum universal site memory size of 64K bytes. By using the iSBC 341 MULTIMODULE board, a maximum of 192K bytes of universal site memory is available. This provides sufficient memory space for on-board network or resource management software.

On-Board DMA

Seven channels of Direct Memory Access (DMA) are provided between serial I/O and on-board dual port RAM by two 8237-5 components. Each of channels 0, 1, 2, 3, 5, 6, and 7 is supported by their own DMA line. Serial channels 0 and 1 are configurable for full duplex DMA. Configuring the full duplex DMA option for Channels 0 and 1 would require Channels 2 and 3 to be interrupt driven or polled. Channel 4 is interrupt driven or polled only.

Two DMA channels are integrated in the 80188 processor. These additional channels can be connected to the iSBX interfaces to provide DMA capability to iSBX MULTIMODULE boards such as the iSBX 218A Floppy Disk Controller MULTIMODULE board.

OPERATING SYSTEM SUPPORT

Intel offers run-time foundation software to support applications that range from general purpose to high-performance solutions.

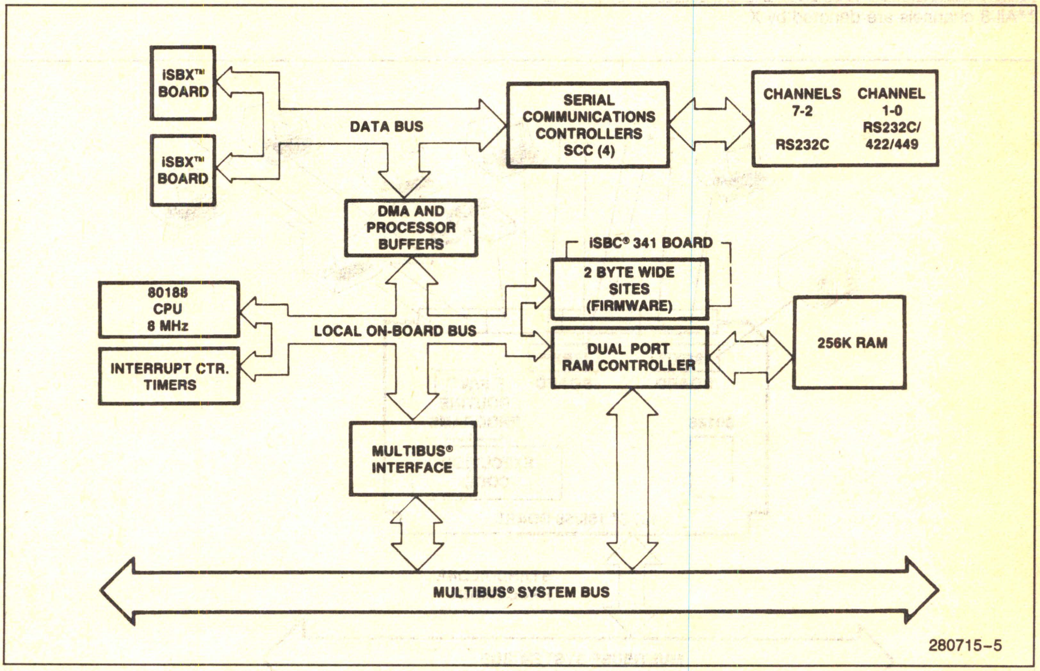


Figure 4. Block Diagram of iSBC® 188/56 Board

Release 6 of the iRMX 86 Operating System provides a rich set of features and options to support sophisticated stand-alone communications applications on the iSBC 188/56 Advanced Communicating Computer. In addition to supporting real-time requirements, the iRMX 86 Operating System Release 6 has a powerful, yet easy to use human interface. Services provided by the iRMX 86 Operating System include facilities for executing programs concurrently, sharing resources and information, servicing asynchronous events and interactively controlling system resources and utilities. The iRMX 86 Operating System is readily extended to support assembler, PL/M, PASCAL, and FORTRAN software development environments. The modular building block software lends itself well to customized application solutions. If the iSBC 188/56 board is acting as an intelligent slave in a system environment, an iRMX 86 driver resident in the host CPU can be written by following the examples in the manual "Guide to Writing Device Driven for iRMX 86 and iRMX 88 I/O Systems".

The iSDM™ 86 System Debug Monitor supports target system debugging for the iSBC 188/56 Advanced Communicating COMMputer board. The monitor contains the necessary hardware, software and documentation required to interface the iSBC 188/56 target system to an Intel microcomputer development system for debugging application software.

The XENIX* 286 Operating System, Release 3, is a fully licensed adaptation of the Bell Laboratories System III UNIX* Operating System. The XENIX system is an interactive, protected, multi-user, multitasking operating system with a powerful, flexible human interface. Release 3 of XENIX 286 includes a software driver for the iSBC 188/56 board (and up to two iSBX 354 MULTIMODULE Boards) acting as an intelligent slave for multi-user applications requiring multiple persons running independent, terminal-oriented jobs. Example applications include distributed data processing, business data processing, software development and engineering or scientific data analysis. XENIX 286 Release 3 Operating System services include device independent I/O, tree-structured file directory and task hierarchies, re-entrant/shared code and system accounting and security access protection.

Table 2. Features of the iSBC® 188/56 Firmware

Feature	Description
Asynchronous Serial Channel Support	Supports the serial channels in asynchronous ASCII mode. Parameters such as baud rate, parity generation, parity checking and character length can be programmed independently for each channel.
Block Data Transfer (On Output)	Relieves the host CPU of character-at-a-time interrupt processing. The iSBC 188/56 board accepts blocks of data for transmission and interrupts the processor only when the entire block is transmitted.
Limited Modem Control	Provides software control of the Data Terminal Ready (DTR) line on all channels. Transitions on the Carrier Detect (CD) line are sensed and reported to the host CPU.
Tandem Modem Support	Transmits an XOFF character when the number of characters in its receive buffer exceeds a threshold value and transmits an XON character when the buffer drains below some other threshold.
Download and Execute Capability	Provides a capability for the host CPU to load code anywhere in the address space of the iSBC 188/56 board and to start executing at any address in its address space.
Power Up Confidence Tests	On board reset, the firmware executes a series of simple tests to establish that crucial components on the board are functional.

FIRMWARE

The iSBC 188/56 Communicating COMMputer board is supplied with resident firmware that supports up to 12 RS232C asynchronous serial channels. In addition, the firmware provides a facility for a host CPU to download and execute code on the iSBC 188/56 board. Simple power-up confidence tests are also included to provide a quick diagnostic service. The firmware converts the iSBC 188/56 COMMputer board to a slave communications controller. As a slave communications controller, it requires a separate MULTIBUS host CPU board and requires the use of MULTIBUS interrupt line to signal the host processor. Table 2 summarizes the features of the firmware.

INTERRUPT CAPABILITY

The iSBC 188/56 board has two programmable interrupt controllers (PICs). One is integrated into the 80188 processor and the other in the 80130 component. The two controllers are configured with the 80130 controller as the master and the 80188 controller as the slave. Two of the 80130 interrupt inputs are connected to the 82530 serial controller components to provide vector interrupt capabilities by the serial controllers. The iSBC 188/56 board provides 22 interrupt levels. The highest level is the NMI (Non-Maskable Interrupt) line which is directly tied to the 80188 CPU. This interrupt is typically used for signaling catastrophic events (e.g. power failure). There are 5 levels of interrupts internal to the 80188 processor. Another 8 levels of interrupts are available from the 80130 component. Of these 8, one is tied to the programmable interrupt controller (PIC) of the 80188 CPU. An additional 8 levels of interrupts are available at the MULTIBUS interface. The iSBC 188/56 board does not support bus vectored interrupts. Table 3 lists the possible interrupt sources.

Table 3. Interrupt Request Sources

Device	Function	Number of Interrupts
MULTIBUS Interface INT0–INT7	Requests from MULTIBUS resident peripherals or other CPU boards.	8
82530 Serial Controllers	Transmit buffer empty, receive buffer full and channel errors 1 and external status.	8 per 82530 Total = 32
Internal 80188 Timer and DMA	Timer 0, 1, 2 outputs and 2 DMA channel interrupts.	5
80130 Timer Outputs	Timer 0, 1, 2 outputs of 80130.	3
Interrupt from Flag Byte Logic	Flag byte interrupt set by MULTIBUS master (through MULTIBUS® I/O Write).	1
Bus Flag Interrupt	Interrupt to MULTIBUS® (Selectable for INT0 to INT7) generated from on-board 80188 I/O Write.	1
iSBX Connectors iSBX DMA	Function determined by iSBX MULTIMODULE board. DMA interrupt from iSBX (TDMA).	4 (Two per Connector) 2
Bus Fail-Safe Timeout Interrupt.	Indicates iSBC 188/48 board timed out either waiting for MULTIBUS access or timed out from no acknowledge while on MULTIBUS System Bus.	1
Latched Interrupt	Converts pulsed event to a level interrupt. Example: 8237A-5 EOP.	1
OR-Gate Matrix	Concentrates up to 4 interrupts to 1 interrupt (selectable by stake pins).	1
Ring Indicator Interrupt	Latches a ring indicator event from serial channels 4, 5, 6, or 7.	1
NOR-Gate Matrix	Inverts up to 2 interrupts into 1 (selectable by stake pins).	1

SUPPORT FOR THE 80130 COMPONENT

Intel does not support the direct processor execution of the iRMX nucleus primitives from the 80130 component. The 80130 component provides timers and interrupt controllers.

EXPANSION

EPROM Expansion

Memory may be expanded by adding Intel compatible memory expansion boards. The universal site memory can be expanded to six sockets by adding the iSBC 341 MULTIMODULE board for a maximum total of 192K bytes of universal site memory.

iSBX™ MULTIMODULE™ Expansion Module

Two 8-bit iSBX MULTIMODULE connectors are provided on the iSBC 188/56 board. Using iSBX modules additional functions can be added to extend the I/O capability of the board. In addition to specialized or custom designed iSBX boards, there is a broad range of iSBX MULTIMODULE boards from the Intel including parallel I/O, analog I/O, IEEE 488 GPIB, floppy disk, magnetic bubbles, video and serial I/O boards.

The serial I/O MULTIMODULE boards available include the iSBX 354 Dual Channel Expansion MULTIMODULE board. Each iSBX 354 MULTIMODULE board adds two channels of serial I/O to the iSBC 188/56 board for a maximum of twelve serial channels. The 82530 serial communications controller on the MULTIMODULE board handles a large variety of serial communications protocols. This is the same serial controller as is used on the iSBC 188/56 board to offer directly compatible expansion capability for the iSBC 188/56 COMMputer board.

MULTIBUS® INTERFACE

The iSBC 188/56 Advanced COMMputer board can be a MULTIBUS master or intelligent slave in a multimaster system. The iSBC 188/56 board incorporates a flag byte signalling mechanism for use in multiprocessor environments where the iSBC 188/56 board is acting as an intelligent slave. The mechanism provides an interrupt handshake from the MULTIBUS System Bus to the on-board-processor and vice-versa.

The Multimaster capabilities of the iSBC 188/56 board offers easy expansion of processing capacity and the benefits of multiprocessing. Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards.

SPECIFICATIONS

Word Size

Instruction—8, 16, 24 or 32 bits
Data Path—8 bits

Processor Clock	82530 Clock	DMA Clock
8 MHz	4.9152 MHz	4 MHz

Dual Port RAM

iSBC 188/56 Board—256 bytes

As viewed from the 80188—64K bytes

As viewed from the MULTIBUS System Bus—Choice: 0, 16K or 48K

EPROM

iSBC® 188/56 Board Using:	Size	On Board Capacity	Address Range
2732	4K	8K bytes	FE000–FFFFF$_H$
2764	8K	16K bytes	FC000–FFFFF$_H$
27128	16K	32K bytes	F8000–FFFFF$_H$
27256	32K	64K bytes	F0000–FFFFF$_H$
27512	64K	128K bytes	E0000–FFFFF$_H$

Memory Expansion

EPROM with iSBC® 341 Board Using:	Capacity	Address Range
2732	24K bytes	F8000–FFFFF$_H$
2764	48K bytes	F0000–FFFFF$_H$
27128	96K bytes	E0000–FFFFF$_H$
27256	192K bytes	C0000–FFFFF$_H$

I/O Capacity

Serial—8 programmable lines using four 82530 components

iSBX MULTIMODULE—2 iSBX single-wide boards

Serial Communications Characteristics

Synchronous—Internal or external character synchronization on one or two synchronous characters

Asynchronous—5–8 bits and 1, 1½, or 2 stop bits per character; programmable clock factor; break detection and generation; parity, overrun, and framing error detection.

Baud Rates

Baud Rate	82530 Count Value (Decimal)
Synchronous X1 Clock	
64000	36
48000	49
19200	126
9600	254
4800	510
2400	1022
1800	1363
1200	2046
300	8190
Asynchronous X16 Clock	
19200	6
9600	14
4800	30
2400	62
1800	83
1200	126
300	510
110	1394

Interfaces

iSBX™ BUS

The iSBC 188/56 board meets iSBX compliance level D8/8 DMA

MULTIBUS® SYSTEM BUS

The iSBC 188/56 board meets MULTIBUS compliance level Master/Slave D8 M24 I16 VO EL.

SERIAL RS232C SIGNALS

CD	Carrier
CTS	Clear to Send
DSR	Data Set Ready
DTE TXC	Transmit Clock
DTR	Data Terminal Ready
RTS	Request to Send
RXC	Receive Clock
RXD	Receive Data
SG	Signal Ground
TXD	Transmit Data
RI	Ring Indicator

RS422A/449 SIGNALS

RC	Receive Common
RD	Receive Data
RT	Receive Timing
SD	Send Data
TT	Terminal Timing

Environmental Characteristics

Temperature: 0 to 55°C at 200 Linear Feet/Min. (LFM) Air Velocity

Humidity: to 90%, non-condensing (25°C to 70°C)

Physical Characteristics

Width: 30.48 cm (12.00 in)
Length: 17.15 cm (6.75 in)
Height: 1.04 cm (0.41 in)
Weight: 595 gm (21 oz)

Electrical Characteristics

The power required per voltage for the iSBC 188/56 board is shown below. These numbers do not include the current required by universal memory sites or expansion modules.

Voltage (Volts)	Current (Amps) typ.	Power (Watts) typ.
+5	4.56A	22.8W
+12	0.12A	1.5W
−12	0.11A	1.3W

Reference Manual

iSBC 188/56 Advanced Data Communications Computer Reference Manual Order Number 148209-001.

ORDERING INFORMATION

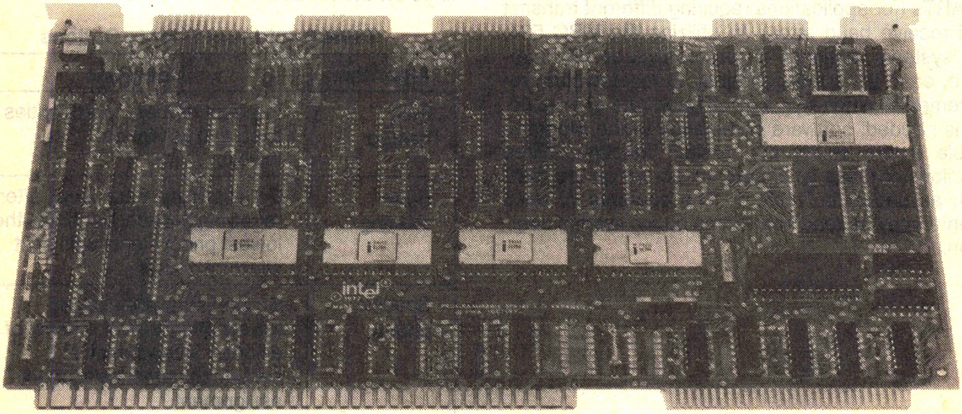
Part Number	Description
iSBC 188/56	8-Serial Channel Advanced Communicating Computer

intel®

iSBC® 534
FOUR CHANNEL COMMUNICATION EXPANSION BOARD

- **Serial I/O Expansion Through Four Programmable Synchronous and Asynchronous Communications Channels**
- **Individual Software Programmable Baud Rate Generation for Each Serial I/O Channel**
- **Two Independent Progammable 16-Bit Interval Timers**
- **Sixteen Maskable Interrupt Request Lines with Priority Encoded and Programmable Interrupt Algorithms**
- **Jumper Selectable Interface Register Addresses**
- **16-Bit Parallel I/O Interface Compatible with Bell 801 Automatic Calling Unit**
- **RS232C/CCITT V.24 Interfaces Plus 20 mA Optically Isolated Current Loop Interfaces (Sockets)**
- **Programmable Digital Loopback for Diagnostics**
- **Interface Control for Auto Answer and Auto Originate Modems**

The iSBC 534 Four Channel Communication Expansion Board is a member of Intel's complete line of memory and I/O expansion boards. The iSBC 534 interfaces directly to any single board computer via the MULTIBUS to provide expansion of system serial communications capability. Four fully programmable synchronous and asynchronous serial channels with RS232C buffering and provision for 20 mA optically isolated current loop buffering are provided. Baud rates, data formats, and interrupt priorities for each channel are individually software selectable. In addition to the extensive complement of EIA Standard RS232C signals provided, the iSBC 534 provides 16 lines of RS232C buffered programmable parallel I/O. This interface is configured to be directly compatible with the Bell Model 801 automatic calling unit. These capabilities provide a flexible and easy means for interfacing Intel iSBC based systems to RS232C and optically isolated current loop compatible terminals, cassettes, asynchronous and synchronous modems, and distributed processing networks.



280238–1

FUNCTIONAL DESCRIPTION

Communications Interface

Four programmable communications interfaces using Intel's 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) are contained on the board.* Each USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bisync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. Each 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in each USART. Each set of RS232C command lines, serial data lines, and signal ground lines are brought out to 26-pin edge connectors that mate with RS232C flat or round cables.

16-Bit Interval Timers

The iSBC 534 provides six fully programmable and independent BCD and binary 16-bit interval timers utilizing two Intel 8253 programmable interval timers.* Four timers are available to the systems designer to generate baud rates for the USARTs under software control. Routing for the outputs from the other two counters is jumper selectable. Each may be independently routed to the programmable interrupt controller to provide real time clocking or to the USARTs (for applications requiring different transmit and receive baud rates). In utilizing the iSBC 534, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given baud rate or time delay is needed, software commands to the programmable timers select the desired function. Three functions of these timers are supported on the iSBC 534, as shown in Table 1. The contents of each counter may be read at any time during system operation.

Table 1. Programmable Timer Functions

Function	Operation
Interrupt on terminal count	When terminal count is reached an interrupt request is generated. This function is used for the generation of real-time clocks.
Rate generator	Divide by N counter. The output will go low for one input clock cycle and high for N−1 input clock periods.
Square wave rate generator	Output will remain high for one-half the count and low for the other half of the count.

Interrupt Request Lines

Two independent Intel 8259A programmable interrupt controllers (PIC's) provide vectoring for 16 interrupt levels.* As shown in Table 2, a selection of three priority processing algorithms is available to the system designer. The manner in which requests are serviced may thus be configured to match system requirements. Priority assignments may be reconfigured dynamically via software at any time during system operation. Any combination of interrupt levels may be masked through storage, via software, of a single byte to the interrupt mask register of each PIC. Each PIC's interrupt request output line may be jumper selected to drive any of the nine interrupt lines on the MULTIBUS.

Table 2. Interrupt Priority Options

Algorithm	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest.
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.

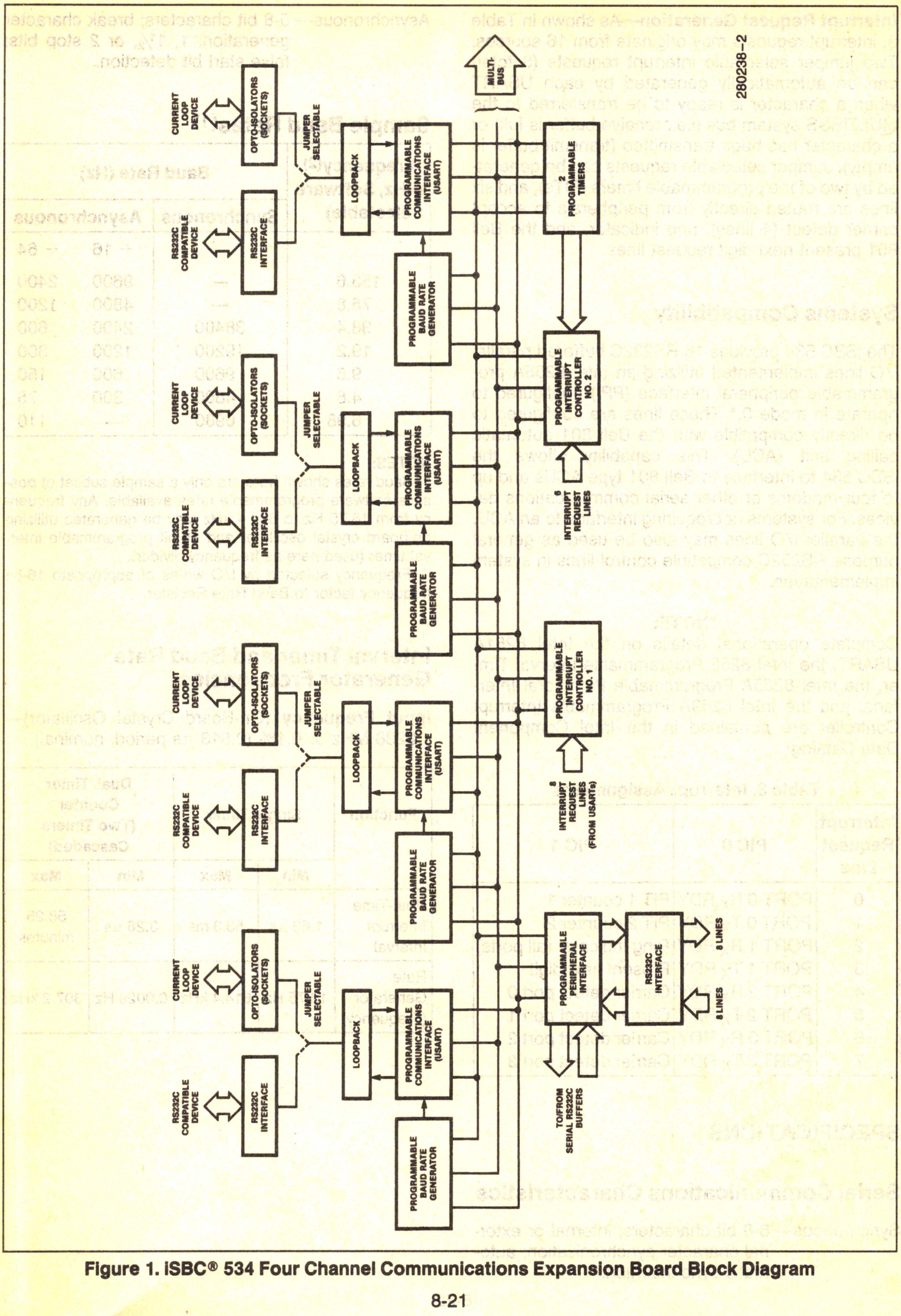


Figure 1. iSBC® 534 Four Channel Communications Expansion Board Block Diagram

Interrupt Request Generation—As shown in Table 3, interrupt requests may originate from 16 sources. Two jumper selectable interrupt requests (8 total) can be automatically generated by each USART when a character is ready to be transferred to the MULTIBUS system bus (i.e., receive buffer is full) or a character has been transmitted (transmit buffer is empty). Jumper selectable requests can be generated by two of the programmable timers (PITs), and six lines are routed directly from peripherals to accept carrier detect (4 lines), ring indicator, and the Bell 801 present next digit request lines.

Systems Compatibility

The iSBC 534 provides 16 RS232C buffered parallel I/O lines implemented utilizing an Intel 8255A programmable peripheral interface (PPI) configured to operate in mode 0.* These lines are configured to be directly compatible with the Bell 801 automatic calling unit (ACU). This capability allows the iSBC 534 to interface to Bell 801 type ACUs and up to four modems or other serial communications devices. For systems not requiring interface to an ACU, the parallel I/O lines may also be used as general purpose RS232C compatible control lines in system implementation.

***NOTE:**
Complete operational details on the Intel 8251A USART, the Intel 8253 Programmable Interval Timer, the Intel 8255A Programmable Peripheral Interface, and the Intel 8259A Programmable Interrupt Controller are contained in the Intel Component Data Catalog.

Table 3. Interrupt Assignments

Interrupt Request Line	PIC 0	PIC 1
0	PORT 0 R_X RDY	PIT 1 counter 1
1	PORT 0 T_X RDY	PIT 2 counter 2
2	PORT 1 R_X RDY	Ring Indicator (all ports)
3	PORT 1 T_X RDY	Present next digit
4	PORT 2 R_X RDY	Carrier detect port 0
5	PORT 2 T_X RDY	Carrier detect port 1
6	PORT 3 R_X RDY	Carrier detect port 2
7	PORT 3 T_X RDY	Carrier detect port 3

SPECIFICATIONS

Serial Communications Characteristics

Synchronous— 5-8 bit characters; internal or external character synchronization; automatic sync insertion.

Asynchronous— 5-8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Sample Baud Rates[1]

Frequency[2] (kHz, Software Selectable)	Baud Rate (Hz)		
	Synchronous	Asynchronous	
		÷ 16	÷ 64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
6.98	6980	—	110

NOTES:
1. Baud rates shown here are only a sample subset of possible software programmable rates available. Any frequency from 18.75 Hz to 614.4 kHz may be generated utilizing on-board crystal oscillator and 16-bit programmable interval timer (used here as frequency divider).
2. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.

Interval Timer and Baud Rate Generator Frequencies

Input Frequency (On-Board Crystal Oscillator)—1.2288 MHz ± 0.1% (0.813 μs period, nominal)

Function	Single Timer		Dual/Timer Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-Time Interrupt Interval	1.63 μs	53.3 ms	3.26 μs	58.25 minutes
Rate Generator (Frequency)	18.75 Hz	614.4 kHz	0.0029 Hz	307.2 kHz

Interfaces—RS232C Interfaces

EIA Standard RS232C Signals provided and supported:

Carrier detect	Receive data
Clear to send	Ring indicator
Data set ready	Secondary receive data
Data terminal ready	Secondary transmit data
Request to send	Transmit clock
Receive clock	Transmit data

Parallel I/O—8 input lines, 8 output lines, all signals RS232C compatible

Bus—All signals MULTIBUS system bus compatible

I/O Addressing

The USART, interval timer, interrupt controller, and parallel interface registers of the iSBC 534 are configured as a block of 16 I/O address locations. The location of this block is jumper selectable to begin at any 16-byte I/O address boundary (i.e., 00H, 10H, 20H, etc.).

I/O Access Time

400 ns	USART registers
400 ns	Parallel I/O registers
400 ns	Interval timer registers
400 ns	Interrupt controller registers

Compatible Connectors

Interface	Pins (qty.)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 2KH43/9 AMK12
Serial and parallel I/O	26	0.1	3m 3462-0001 or TI H312113

Compatible Opto-Isolators

Function	Supplier	Part Number
Driver	Fairchild General Electric Monsanto	4N33
Receiver	Fairchild General Electric Monsanto	4N37

Physical Characteristics

Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)
Weight: 14 oz. (398 gm)

Electrical Characteristics

Average DC Current

Voltage	Without Opto-Isolators	With Opto-Isolators(1)
$V_{CC} = +5V$	1.9 A, max	1.9 A, max
$V_{DD} = +12V$	275 mA, max	420 mA, max
$V_{AA} = -12V$	250 mA, max	400 mA, max

NOTE:
1. With four 4N33 and four 4N37 opto-isolator packages installed in sockets provided to implement four 20 mA current loop interfaces.

Environmental Characteristics

Operating Temperature: 0°C to +55°C

Reference Manual

502140-002—iSBC 534 Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

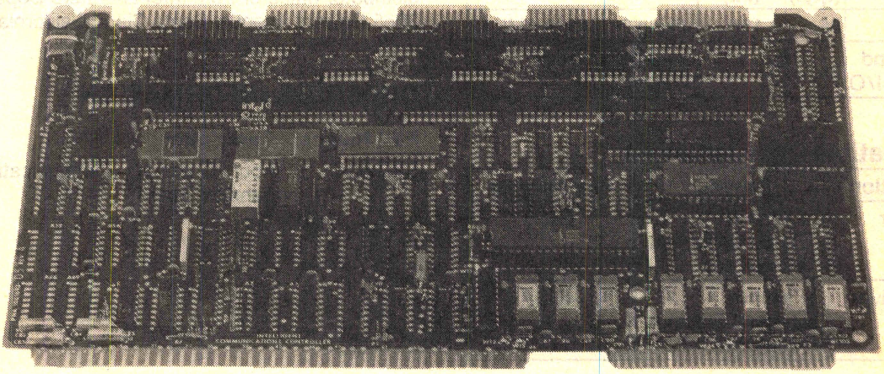
ORDERING INFORMATION

Part Number	Description
SBC 534	Four Channel Communication Expansion Board

iSBC® 544
INTELLIGENT COMMUNICATIONS CONTROLLER

- **iSBC® Communications Controller Acting as a Single Board Communications Computer or an Intelligent Slave for Communications Expansion**
- **On-Board Dedicated 8085A Microprocessor Providing Communications Control and Buffer Management for Four Programmable Synchronous/Asynchronous Channels**
- **Sockets for Up To 8K Bytes of EPROM**
- **16K Bytes of Dual Port Dynamic Read/Write Memory with On-Board Refresh**
- **Extended MULTIBUS® Addressing Permits iSBC 544 Board Partitioning into 16K-Byte Segments in a 1-Megabyte Address Space**
- **Ten Programmable Parallel I/O Lines Compatible with Bell 801 Automatic Calling Unit**
- **Twelve Levels of Programmable Interrupt Control**
- **Individual Software Programmable Baud Rate Generation for Each Serial I/O Channel**
- **Three Independent Programmable Interval Timer/Counters**
- **Interface Control for Auto Answer and Auto Originate Modem**

The iSBC 544 Intelligent Communications Controller is a member of Intel's family of single-board computers, memory, I/O, and peripheral controller boards. The iSBC 544 board is a complete communications controller on a single 6.75 x 12.00 inch printed circuit card. The on-board 8085A CPU may perform local communications processing by directly interfacing with on-board read/write memory, nonvolatile read only memory, four synchronous/asynchronous serial I/O ports, RS232/RS366 compatible parallel I/O, programmable timers, and programmable interrupts.



280239–1

November 1986
Order Number: 280239-001

FUNCTIONAL DESCRIPTION

Intelligent Communications Controller

Two Mode Operation — The iSBC 544 board is capable of operating in one of two modes: 1) as a single board communications computer with all computer and communications interface hardware on a single board; 2) as an "intelligent bus slave" that can perform communications related tasks as a peripheral processor to one or more bus masters. The iSBC 544 may be configured to operate as a standalone single board communications computer with all MPU, memory and I/O elements on a single board. In this mode of operation, the iSBC 544 may also interface with expansion memory and I/O boards (but no additional bus masters). The iSBC 544 performs as an intelligent slave to the bus master by performing all communications related tasks. Complete synchronous and asynchronous I/O and data management are controlled by the on-board 8085A CPU to coordinate up to four serial channels. Using the iSBC 544 as an intelligent slave, multichannel serial transfers can be managed entirely on-board, freeing the bus master to perform other system functions.

Architecture — The iSBC 544 board is functionally partitioned into three major sections: I/O, central computer, and shared dual port RAM memory (Figure 1). The I/O hardware is centered around the four Intel 8251A USART devices providing fully programmable serial interfacing. Included here as well is a 10-bit parallel interface compatible with the Bell 801 automatic calling unit, or equivalent. The I/O is under full control of the on-board CPU and is protected from access by system bus masters. The second major segment of the intelligent communications controller is a central computer, with an 8085A CPU providing powerful processing capability. The 8085A together with on-board EPROM/ROM, static RAM, programmable timers/counters, and programmable

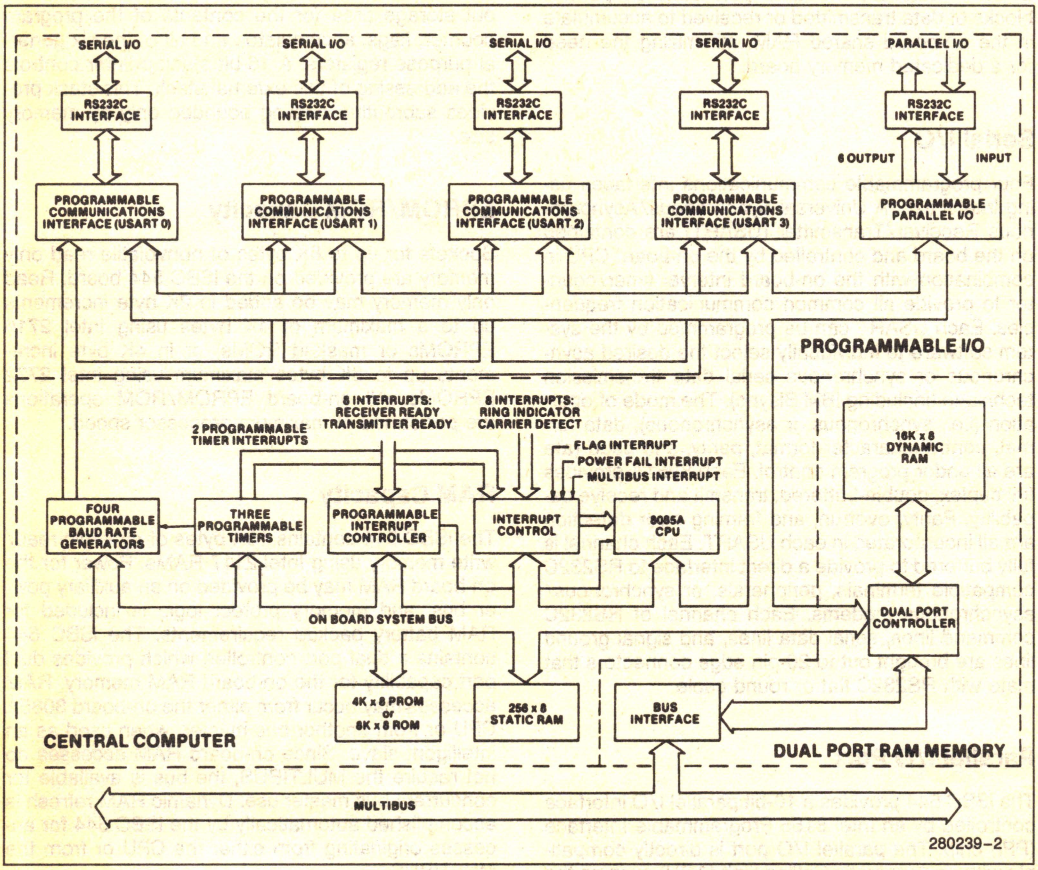


Figure 1. iSBC® 544 Intelligent Communications Controller Block Diagram

interrupt control provide the intelligence to manage sophisticated communications operations on-board the iSBC 544 board. The timer/counters and interrupt control are also common to the I/O area providing programmable baud rates to the USARTs and prioritizing interrupts generated from the USARTs. The central computer functions are protected for access only by the on-board 8085A. Likewise, the on-board 8085A may not gain access to the system bus when being used as an intelligent slave. When the iSBC 544 is used as a bus master, the on-board 8085A CPU controls complete system operation accessing on-board functions as well as memory and I/O expansion. The third major segment, dual port RAM memory, is the key link between the iSBC 544 intelligent slave and bus masters managing the system functions. The dual port concept allows a common block of dynamic memory to be accessed by the on-board 8085A CPU and off-board bus masters. The system program can, therefore, utilize the shared dual port RAM to pass command and status information between the bus masters and on-board CPU. In addition, the dual port concept permits blocks of data transmitted or received to accumulate in the on-board shared RAM, minimizing the need for a dedicated memory board.

Serial I/O

Four programmable communications interfaces using Intel's 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) are contained on the board and controlled by the on-board CPU in combination with the on-board interval timer/counter to provide all common communication frequencies. Each USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bisync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. Each 8251A provides full duplex, double-buffered, transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in each USART. Each channel is fully buffered to provide a direct interface to RS232C compatible terminals, peripherals, or synchronous/asynchronous modems. Each channel of RS232C command lines, serial data lines, and signal ground lines are brought out to 26-pin edge connectors that mate with RS232C flat or round cable.

Parallel I/O Port

The iSBC 544 provides a 10-bit parallel I/O interface controlled by an Intel 8155 Programmable Interface (PPI) chip. The parallel I/O port is directly compatible with an Automatic Calling Unit (ACU) such as the Bell Model 801, or equivalent, and can also be used for auxiliary functions. All signals are RS232C compatible, and the interface cable signed assignments meet RS366 specifications. For systems not requiring an ACU interface, the parallel I/O port can be used for any general purpose interface requiring RS232C compatibility.

Central Processing Unit

Intel's powerful 8-bit n-channel 8085A CPU, fabricated on a single LSI chip, is the central processor for the iSBC 544. The 8085A CPU is directly software compatible with the Intel 8080A CPU. The 8085A contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. The minimum instruction execution time is 1.45 microseconds. The 8085A CPU has a 16-bit program counter. An external stack, located within any portion of iSBC 544 read/write memory, may be used as a last-in/first-out storage area for the contents of the program counter, flags, accumulator, and all of the six general purpose registers. A 16-bit stack pointer controls the addressing of this external stack. This stack provides subroutine nesting bounded only by memory size.

EPROM/ROM Capacity

Sockets for up to 8K bytes of nonvolatile read only memory are provided on the iSBC 544 board. Read only memory may be added in 2K byte increments up to a maximum of 4K bytes using Intel 2716 EPROMs or masked ROMs; or in 4K byte increments up to 8K bytes maximum using Intel 2732 EPROMs. All on-board EPROM/ROM operations are performed at maximum processor speed.

RAM Capacity

The iSBC 544 contains 16K bytes of dynamic read/write memory using Intel 2117 RAMs. Power for the on-board RAM may be provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements. The iSBC 544 contains a dual port controller, which provides dual port capability for the on-board RAM memory. RAM accesses may occur from either the on-board 8085A CPU or from another bus master, when used as an intelligent slave. Since on-board RAM accesses do not require the MULTIBUS, the bus is available for concurrent bus master use. Dynamic RAM refresh is accomplished automatically by the iSBC 544 for accesses originating from either the CPU or from the MULTIBUS.

Addressing — On board RAM, as seen by the on-board 8085A CPU, resides at address 8000_H–$BFFF_H$. On-board RAM, as seen by an off-board CPU, may be placed on any 4K byte address boundary. The iSBC 544 provides extended addressing jumpers to allow the on-board RAM to reside within a one megabyte address space when accessed via the MULTIBUS. In addition, jumper options are provided which allow the user to protect 8K or 12K bytes on-board RAM for use by the on-board 8085 CPU only. This reserved RAM space is not accessible via the MULTIBUS and does not occupy any system address space.

Static RAM — The iSBC 544 board also has 256 bytes of static RAM located on the Intel 8155 PPI. This memory is only accessible to the on-board 8085A CPU and is located at address $7F00_H$–$7FFF_H$.

Programmable Timers

The iSBC 544 board provides seven fully programmable and independent interval timer/counters utilizing two Intel 8253 Programmable Interval Timers (PIT), and the Intel 8155. The two Intel 8253 PITs provide six independent BCD or binary 16-bit interval timer/counters and the 8155 provides one 14-bit binary timer/counter. Four of the PIT timers (BDG0–3) are dedicated to the USARTs providing fully independent programmable baud rates.

Three General Use Timers — The fifth timer (BDG4) may be used as an auxiliary baud rate to any of the four USARTs or may alternatively be cascaded with timer six to provide extended interrupt intervals. The sixth PIT timer/counter (TINT1) can be used to generate interrupt intervals to the on-board 8085A. In addition to the timer/counters on the 8253 PITs, the iSBC 544 has a 14-bit timer available on the 8155 PPI providing a third general use timer/counter (TINT0). This timer output is jumper selectable to the interrupt structure of the on-board 8085A CPU to provide additional timer/counter capability.

Timer Functions — In utilizing the iSBC 544 board, the systems designer simply configures, via software, each timer independently to meet systems requirements. Whenever a given baud rate or interrupt interval is needed, software commands to the programmable timers select the desired function. The on-board PITs together with the 8155 provide a total of seven timer/counters and six operating modes. Mode 3 of the 8253 is the primary operating mode of the four dedicated USART baud rate generators. The timer/counters and useful modes of operation for the general use timer/counters are shown in Table 1.

Interrupt Capability

The iSBC 544 board provides interrupt service for up to 21 interrupt sources. Any of the 21 sources may interrupt the intelligent controller, and all are brought through the interrupt logic to 12 interrupt levels. Four interrupt levels are handled directly by the interrupt processing capability of the 8085A CPU and eight levels are serviced from an Intel 8259A Programmable Interrupt Controller (PIC) routing an interrupt request output to the INTR input of the 8085A (see Table 2).

Table 1. Programmable Timer Functions

Function	Operation	Counter
Interrupt on Terminal Count (Mode 0)	When terminal count is reached, an interrupt request is generated. This function is useful for generation of real-time clocks.	**8253** TINT1
Rate Generator (Mode 2)	Divide by N counter. The output will go low for one input clock cycle and high for N − 1 input clock periods.	**8253** BDG4*
Square-Wave Rate Generator (Mode 3)	Output will remain high until one-half the TC has been completed, and go low for the other half of the count. This is the primary operating mode used for generating a Baud rate clocked to the USARTs.	**8253** BDG0–4 TINT1
Software Triggered Strobe (Mode 4)	When the TC is loaded, the counter will begin. On TC the output will go low for one input clock period.	**8253** BDG4* TINT1
Single Pulse	Single pulse when TC reached.	**8155** TINT0
Repetitive Single Pulse	Repetitive single pulse each time TC is reached until a new command is loaded.	**8155** TINT0

* BDG4 is jumper selectable as an auxiliary baud rate generator to the USARTs or as a cascaded output to TINT1. BDG4 may be used in modes 2 and 4 only when configured as a cascaded output.

Table 2. Interrupt Vector Memory Locations

Interrupt Source		Vector Location	Interrupt Level
Power Fail	TRAP	24_H	1
8253 TINT1	RST 7.5	$3C_H$	2
8155 TINTO			
Ring Indicator[1]	RST 6.5	34_H	3
Carrier Detect			
Flag Interrupt	RST 5.5	$2C_H$	4
INTO/-INT7/ (1 of 8)			
RXRDY0	INTR	Programmable	5–12
TXRDY0			
RXRDY1			
TXRDY1			
RXRDY2			
TXRDY2			
RXRDY3			
TXRDY3			

NOTE:
1. Four ring indicator interrupts and four carrier detect interrupts are summed to the RST 6.5 input. The 8155 may be interrogated to inspect any one of the eight signals.

Interrupt Sources — The 22 interrupt sources originate from both on-board communications functions and the MULTIBUS. Two interrupts are routed from each of the four USARTs (8 interrupts total) to indicate that the transmitter and receiver are ready to move a data byte to or from the on-board CPU. The PIC is dedicated to accepting these 8 interrupts to optimize USART service request. One of eight interrupt request lines are jumper selectable for direct interface from a bus master via the system bus. Two auxiliary timers (TINTO from 8155 and TINT1 from 8253) are jumper selectable to provide general purpose counter/timer interrupts. A jumper selectable Flag Interrupt is generated to allow any bus master to interrupt the iSBC 544 by writing into the base address of the shared dual port memory accessable to the system. The Flag Interrupt is then cleared by the iSBC 544 when the on-board processor reads the base address. This interrupt provides an interrupt link between a bus master and intelligent slave (see System Programming). Eight inputs from the serial ports are monitored to detect a ring indicator and carrier detect from each of the four channels. These eight interrupt sources are summed to a single interrupt level of the 8085A CPU. If one of these eight interrupts occur, the 8155 PPI can then be interrogated to determine which port caused the interrupt. Finally, a jumper selectable Power Fail Interrupt is available from the MULTIBUS to detect a power down condition.

8085 Interrupt — Thirteen of the twenty-two interrupt sources are available directly to four interrupt inputs of the on-board 8085A CPU. Requests routed to the 8085A interrupt inputs, TRAP, RST 7.5, RST 6.5 and RST 5.5 have a unique vector memory address. An 8085A jump instruction at each of these addresses then provides software linkage to interrupt service routines located independently anywhere in the Memory. All interrupt inputs with the exception of the TRAP may be masked via software.

8259A Interrupts — Eight interrupt sources signaling transmitter and receiver ready from the four USARTs are channeled directly to the Intel 8259A PIC. The PIC then provides vectoring for the next eight interrupt levels. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts transmitter and receiver interrupts from the four USARTs. It then determines which of the incoming requests is of highest priority, determines whether this request is of higher priority than the level currently being serviced, and , if appropriate, issues an interrupt to the CPU. The output of the PIC is applied directly to the INTR input of the 8085A. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. When the 8085A responds to a PIC interrupt, the PIC will generate a CALL instruction for each interrupt level. These addresses are equally spaced at intervals of 4 or 8 (software selectable) bytes. Interrupt response to the PIC is software programmable to a 32- or 64-byte block of memory. Interrupt sequences may be expanded from this block with a single 8085A jump instruction at each of these addresses.

Interrupt Output — In addition, the iSBC 544 board may be jumper selected to generate an interrupt from the on-board serial output data (SOD) of the 8085A. The SOD signal may be jumpered to any one of the 8 MULTIBUS interrupt lines (INTO/-INT7/) to provide an interrupt signal directly to a bus master.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the iSBC 635 Power Supply or equivalent.

Expansion Capabilities

When the iSBC 544 board is used as a single board communications controller, memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS™ compatible expansion boards. In this mode, no other bus masters may be configured in the system. Memory may be expanded to a 65K byte capacity by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expan-

sion boards. Furthermore, multiple iSBC 544 boards may be included in an expanded system using one iSBC 544 board as a single board communications computer and additional controllers as intelligent slaves.

System Programming

In the system programming environment, the iSBC 544 board appears as an additional RAM memory module when used as an intelligent slave. The master CPU communicates with the iSBC 544 board as if it were just an extension of system memory. Because the iSBC 544 board is treated as memory by the system, the user is able to program into it a command structure which will allow the iSBC 544 board to control its own I/O and memory operation. To enhance the programming of the iSBC 544 board, the user has been given some specific tools. The tools are: 1) the flag interrupt, 2) an on-board RAM memory area that is accessible to both an off-board CPU and the on-board 8085A through which a communications path can exist, and 3) access to the bus interrupt line.

Flag Interrupt — The Flag Interrupt is generated anytime a write command is performed by an off-board CPU to the base address of the iSBC 544 board's RAM. This interrupt provides a means for the master CPU to notify the iSBC 544 board that it wishes to establish a communications sequence. In systems with more than one intelligent slave, the flag interrupt provides a unique interrupt to each slave outside the normal eight MULTIBUS interrupt lines (INT0/-INT7/).

On-Board RAM — The on-board 16K byte RAM area that is accessible to both an off-board CPU and the on-board 8085A can be located on any 4K boundary in the system. The selected base address of the iSBC 544 RAM will cause an interrupt when written into by an off-board CPU.

Bus Access — The third tool to improve system operation as an intelligent slave is access to the MULTIBUS interrupt lines. The iSBC 544 board can both respond to interrupt signals from an off-board CPU, and generate an interrupt to the off-board CPU via the MULTIBUS.

System Development Capability

The development cycle of iSBC 544 board based products may be significantly reduced using the Intellec series microcomputer development systems. The Intellec resident macroassembler, text editor, and system monitor greatly simplify the design, development and debug of iSBC 544 system software. An optional ISIS-II diskette operating system provides a linker, object code locater, and library manager. A unique in-circuit emulator (ICE-85) option provides the capability of developing and debugging software directly on the iSBC 544 board.

SPECIFICATIONS

Serial Communications Characteristics

Synchronous — 5–8 bit characters; automatic sync insertion; parity.

Asynchronous — 5–8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection; break character detection.

Baud Rates

Frequency (KHz)(1) (Software Selectable)	Baud Rate (Hz)(2)		
	Synchronous	Asynchronous	
		÷16	÷64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
6.98	6980	—	110

NOTES:
1. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.
2. Baud rates shown here are only a sample subset of possible software programmable rates available. Any frequency from 18.75 Hz to 614.4 KHz may be generated utilizing on-board crystal oscillator and 16-bit Programmable Interval Timer (used here as a frequency divider).

8085A CPU

Word Size — 8, 16 or 24 bits/instruction; 8 bits of data

Cycle Time — 1.45/μs ±0.01% for fastest executable instruction; i.e., four clock cycles.

Clock Rate — 2.76 MHz ± 0.1%

System Access Time

Dual port memory — 740 ns

NOTE:
Assumes no refresh contention.

Memory Capacity

On-Board ROM/PROM — 4K, or 8K bytes of user installed ROM or EPROM

On-Board Static RAM — 256 bytes on 8155

On-Board Dynamic RAM (on-board access) — 16K bytes. Integrity maintained during power failure with user-furnished batteries (optional)

On-Board Dynamic RAM (MULTIBUS access) — 4K, 8K, or 16K bytes available to bus by swtich selection

Memory Addressing

On-Board ROM/PROM — 0–0FFF (using 2716 EPROMs or masked ROMs); 0–1FFF (using 2732A EPROMs)

On-Board Static RAM — 256 bytes: 7F00–7FFF

On-Board Dynamic RAM (on-board access) — 16K bytes: 8000–BFFF.

On-Board Dynamic RAM (MULTIBUS® access) — any 4K increment 00000–FF000 which is switch and jumper selectable. 4K, 8K or 16K bytes can be made available to the bus by switch selection.

I/O Capacity

Serial — 4 programmable channels using four 8251A USARTs

Parallel — 10 programmable lines available for Bell 801 ACU, or equivalent use. Two auxiliary jumper selectable signals

I/O Addressing

On-Board Programmable I/O

Port	Data	Control
USART 0	D0	D1
USART 1	D2	D3
USART 2	D4	D5
USART 3	D6	D7
8155 PPI	E9 (Port A) EA (Port B) EB (Port C)	E8

Interrupts

Address for 8259A Registers (Hex notation, I/O address space)

E6	Interrupt request register
E6	In-service register
E7	Mask register
E6	Command register
E7	Block address register
E6	Status (polling register)

NOTE:
Several registers have the same physical address: Sequence of access and one data bit of the control word determines which register will respond.

Interrupt levels routed to the 8085 CPU automatically vector the processor to unique memory locations:

24	TRAP
3C	RST 7.5
34	RST 6.5
2C	RST 5.5

Timers

Addresses for 8253 Registers (Hex notation, I/O address space)

Programmable Interrupt Timer One

D8	Timer 0	BDG0
D9	Timer 1	BDG1
DA	Timer 2	BDG2
DB	Control register	

Programmable Interrupt Timer Two

DC	Timer 0	BDG3
DD	Timer 1	BDG4
DE	Timer 2	TINT1
DF	Control register	

Address for 8155 Programmable Timer

E8	Control	
	Timer (LSB)	TINTO
ED	Timer (MSB)	TINTO

Input Frequencies — Jumper selectable reference 1.2288 MHz ± 0.1% (0.814 μs period nominal) or 1.843 MHz ±0.1% crystal (0.542 μs period, nominal)

Output Frequencies (at 1.2288 MHz)

Function	Single Timer/Counter		Dual Timer/Counter (two timers cascaded)	
	Min	Max	Min	Max
Real-Time Interrupt Interval	1.63 μs	53.3 μs	3.26 μs	58.25 min
Rate Generator (frequency)	18.75 Hz	614.4 KHz	0.00029 Hz	307.2 KHz

Interfaces

Serial I/O — EIA Standard RS232C signals provided and supported:

Carrier Detect	Receiver Data
Clear to Send	Ring Indicator
Data Set Ready	Secondary Receive Data*
Data Terminal Ready	Secondary Transmit Data *
Request to Send	Transmit Clock
Receive Clock	Transmit Data
	DTE Transmit clock

* Optional if parallel I/O port is not used as Automatic Calling Unit.

Parallel I/O — Four inputs and eight outputs (includes two jumper selectable auxiliary outputs). All signals compatible with EIA Standard RS232C. Directly compatible with Bell Model 801 Automatic Calling Unit, or equivalent.

MULTIBUS — Compatible with iSBC MULTIBUS.

On-Board Addressing

All communications to the parallel and serial I/O ports, to the timers, and to the interrupt controller, are via read and write commands from the on-board 8085A CPU.

Auxiliary Power

An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Connectors

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	Viking 2KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000 or AMP 88083-1
Serial I/O	26	0.1	3M 3462-000 or AMP 88373-5

Memory Protect

An active-low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during the system power-down sequences.

Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	15
Commands	Tri-state	32

NOTE:
Used as a master in the single board communications computer mode.

Physical Characteristics

Width:	30.48 cm (12.00 inches)
Depth:	17.15 cm (6.75 inches)
Thickness:	1.27 cm (0.50 inch)
Weight:	3.97 gm (14 ounces)

Electrical Characteristics

DC Power Requirements

Current Requirements				
Configuration	**V_{CC} = +5V ±5% (max)**	**V_{DD} = ±12V ±5% (max)**	**V_{BB} = −5V(3) ±5% (max)**	**V_{AA} = −12V ±5% (max)**
With 4K EPROM (using 2716)	I_{CC} = 3.4A max	I_{DD} = 350 mA max	I_{BB} = 5 mA max	I_{AA} = 200 mA max
Without EPROM	3.3A max	350 mA max	5 mA max	200 mA max
RAM only(1)	390 mA max	176 mA max	5 mA max	—
RAM(2) refresh only	390 mA max	20 mA max	5 mA max	

NOTES:
1. For operational RAM only, for AUX power supply rating.
2. For RAM refresh only. Used for battery backup requirements. No RAM accessed.
3. V_{BB} is normally derived on-board from V_{AA}, eliminating the need for a V_{BB} supply. If it is desired to supply V_{BB} from the bus, the current requirement is as shown.

Environmental Characteristics

Operating Temperature: 0°C to 55°C (32°F to 131°F)

Relative Humidity: To 90% without condensation

Reference Manual

502160 — iSBC 544 Intelligent Communications Controller Board Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

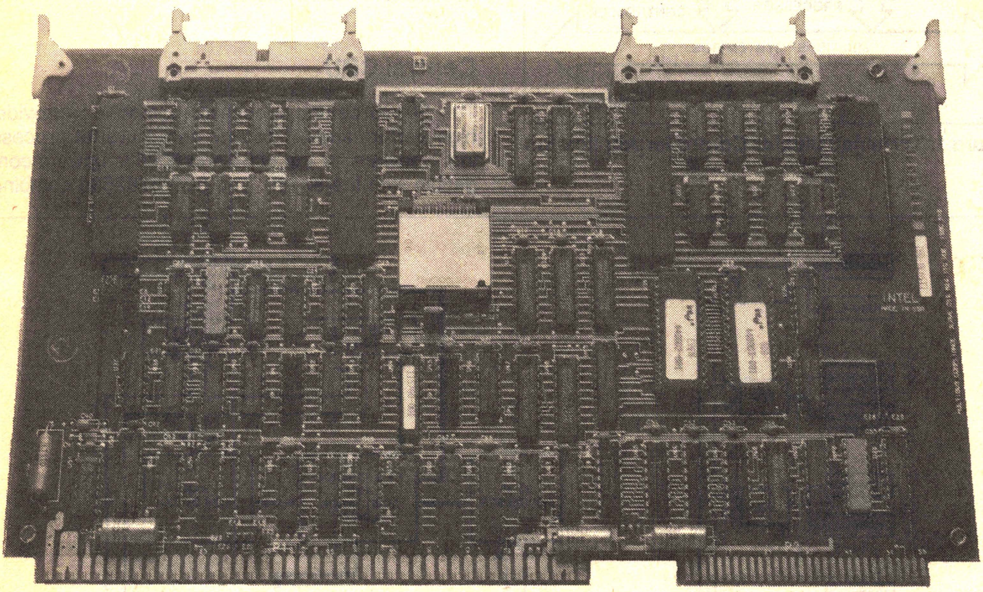
Part Number	**Description**
iSBC 544	Intelligent Communications Controller

intel®

iSBC® 548 HIGH PERFORMANCE TERMINAL CONTROLLER

- Intelligent Slave Commission Board
- 8 Serial RS232 Communication Channels
- 8 MHz 80186 Microprocessor
- Supports Full Duplex Asynchronous Transmissions
- 128K Bytes Zero Wait State DRAM (32K Dual Port)
- Two 28-Pin JEDEC PROM Sites for up to 64K Zero Wait State EPROM
- Individual Software Programmable Baud Rate Generation for Each Channel
- Resident Firmware for Terminal and Modem Control

The iSBC® 548 High Performance Terminal Controller is an intelligent 8-channel single board computer for asynchronous terminal control applications. The iSBC 548 MULTIBUS® I board adds the power of a 8 MHz 80186 microprocessor to the Intel line of OEM communication controllers. Acting as an intelligent slave for communication expansion, this board provides a high performance, low cost solution for multi-user systems.



280250–1

September 1986
Order Number: 280250-001

OPERATING ENVIRONMENT

The iSBC 548 board is designed to be a terminal/cluster controller. A terminal controller offloads the system processor of communication message handling. (See Figure 1.) The dual-port RAM provides an on-board buffer to handle incoming and outgoing messages at data rates up to 19.2K baud. The firmware supplied on the iSBC 548 Communications Controller supports 8 asynchronous RS232 serial channels, provides modem control and performs power-up diagnostics. Each serial channel can be individually programmed for different baud rates to allow system configurations with differing terminal types.

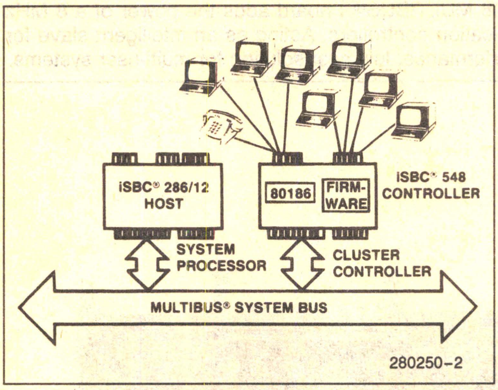


Figure 1. Terminal/Cluster Controller Application

The iSBC 548's high performance CPU provides intelligence to handle communication functions normally assigned to the system CPU, freeing the bus master to perform other system functions. This distribution of intelligence results in optimizing system performance.

ARCHITECTURE

The three major functional areas are Serial I/O, CPU and Memory (see Figure 2.)

Serial I/O

Four 82530 Serial Communications Controllers (SCCs) provide eight channels of half/full duplex serial I/O. Configurability of the 82530 allows the user to configure it to handle all asynchronous data formats regardless of data size, number of start or stop bits, or parity requirements. The synchronous transmission features of the 82530 are not supported. An on-chip baud rate generator allows independent baud rates on each channel. The serial lines can be brought to the back-panel via two 40-pin shell connectors and ribbon cable.

Central CPU

The 80186 central processor component provides high performance, flexibility and powerful processing. Software for the 8088 and 8086 is upward compatible with the 80186. The 80186/82530 combina-

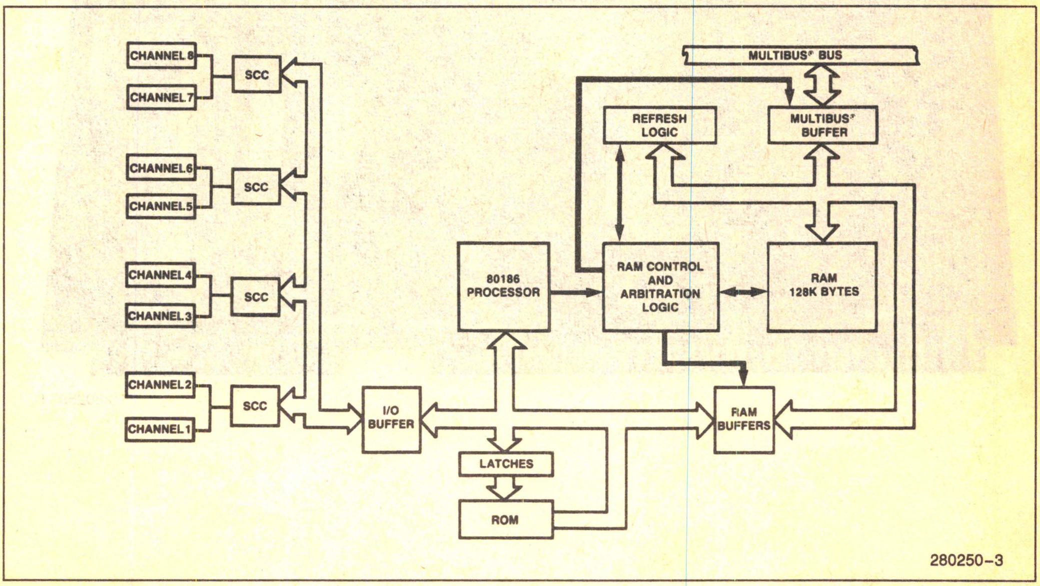


Figure 2. iSBC® 548 Functional Block Diagram

tion with on-board PROM/EPROM sites, and dual-port RAM provide the intelligence and speed to manage multi-user communications.

Memory

There are three areas of memory on-board; private RAM, dual-port RAM and EPROM. The iSBC 548 Communications Controller contains 128K bytes of on-board RAM, 32K bytes of which is dual-port RAM this is addressable by the 80186 on-board. The dual port memory is configurable in a 16M byte address space on 32K byte boundaries as addressed from the MULTIBUS port. The starting address is jumper selectable.

The third area of memory is EPROM memory expansion. Two 28-pin JEDEC sockets are provided. These sockets come populated with two EPROMs which contain the controller firmware. The boards can support 2764, 27128 and 27256 EPROMs, giving a total capacity of 64K bytes. The EPROM runs with zero wait states if EPROMS of access times 250 ns or less are used. No jumper changes are needed to access different size EPROMs.

OPERATING SYSTEM SUPPORT

For those applications needing a real time, multitasking operating system, Intel offers the iRMX™ 86 and iRMX 286 Operating Systems. The iRMX operating systems are particularly well suited for applications where the processor is simultaneously controlling multiple, real-time interrupt-intensive processes. Typical applications include machine and process control, data acquisition, signal processing, and front-end processing.

Intel also offers the XENIX* 286 Operating System which is designed for those applications needing an interactive, multiple user system. Intel's XENIX operating system is a fully licensed derivative of UNIX*, enhanced by Intel to provide device driver support for Intel board and component products plus other features that yeild greater flexibility, increased reliability, and easier configurability. Typical applications include distributed data processing, business data processing, software development and engineering or scientific data analysis.

FIRMWARE

The iSBC 548 High Performance Terminal Controller is supplied with resident firmware that supports 8 RS232C asynchronous serial channels. In addition, power-up confidence tests are also included to provide a quick diagnostic service.

*XENIX is a trademark of Microsoft Corporation
*UNIX is a trademark of Bell Laboratories

Asynchronous Serial Channel Support

Supports the serial channels in asynchronous mode. Parameters such as baud rate, parity generation, parity checking and character length can be programmed independently for each channel.

Limited Modem Control

The firmware provides software control of the Data Terminal Ready (DTR) line on all channels. Transitions on the Carrier Detect (CD) line are sensed and reported to the host CPU. Clear to Send (CTS) and Carrier Detect (CD) can be set to gate iSBC 548 transmissions and receptions respectively. Data Set Ready (DSR) and Ring Indicator (RI) are sensed.

Block Data Transfer

The firmware relieves the host of character-at-a-time interrupt processing. The iSBC 548 board accepts blocks of data for transmissions and interrupts the processor only when the entire block is transmitted.

Download and Execute Facility

Provides a capability for the host CPU to load code in the address space of the iSBC 548 board and for the iSBC 548 board to start executing at any address in its address space.

Power Up Confidence Tests

On board reset, the firmware executes a series of tests to establish that crucial components on the board are functional.

MULTIBUS® INTERFACE

The iSBC 548 Controller Board operates as an intelligent slave using a flag byte signalling mechanism. This mechanism provides an interrupt handshake from the MULTIBUS System Bus to the on-board processor and vice-versa.

SPECIFICATIONS

Serial Communications Characteristics

ASYNCHRONOUS ONLY

6–8 bit character length
1, 1½, or 2 stop bits per character
Parity
Programmable clock
Break generation
Framing error detection

BAUD RATES

The on-board firmware can automatically detect and set baud rates of 150, 300, 600, 1200, 2400, 4800, 9600 and 19200. Other baud rates can be set by the host.

SERIAL RS232C SIGNALS SUPPORTED

CD	Carrier Detect
RXD	Receive Data
TXD	Transmit Data
DTR	Data Terminal Ready
SG	Signal Ground
DSR	Data Set Ready
RTS	Ready to Send
CTS	Clear to Send
RI	Ring Indicator

These signals are supported by the iSBC 548 Controller and on-board firmware. All signals may not be supported by the host operating system.

Memory

On-Board RAM: 128K bytes total

Private RAM: 96K bytes

Dual Port RAM: 32K bytes, can be addressed from MULTIBUS interface at any 32K boundary between 80000H and F80000H and between F80000 and FF80000.

EPROM OPTIONS

Component	On-Board Capacity	Start Address
2764	16K	FC000H
27128	32K	F8000H
27256	64K	F0000H

MULTIBUS® System Bus Interface

The iSBC 548 board meets MULTIBUS (IEEE 796) bus specification D16 M24 I16 VO E.

Environmental Characteristics

Temperature: 0 to 55 degrees Centigrade at 200 Linear Feet/Minute (LFM) Air Velocity

Humidity: 5% to 90%, non-condensing (25 to 70 degrees Centigrade)

Physical Characteristics

Width: 30.48 cm (12.00 in)
Length: 16.87 cm (6.75 in)
Height: 1.27 cm (0.50 in)
Weight: 400 gm (14 oz)

Power Requirements

Maximum Power Required per Voltage		
Voltage (Volts)	**Current (Amps)**	**Power (Watts)**
+5	3.49	17.5
+12	0.14	1.7
−12	0.11	1.3

ORDERING INFORMATION

Part Number	Description
ISBC 548	8 Channel High Performance Terminal Controller

Reference Manuals

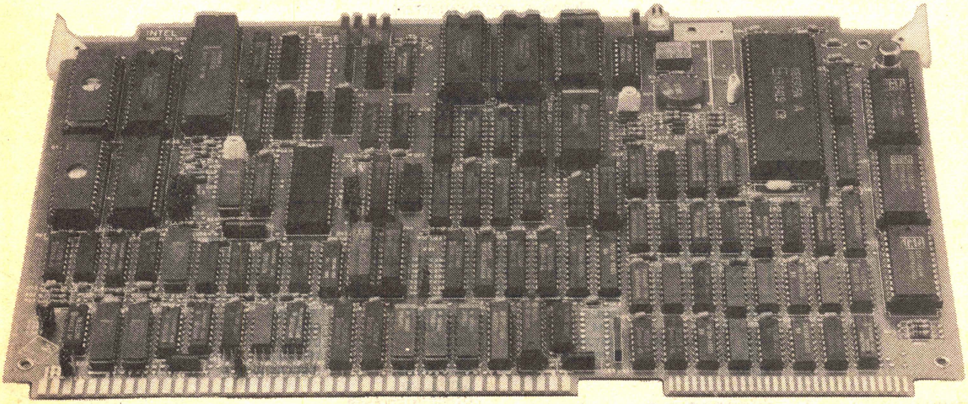
iSBC® 548 High Performance Terminal Controllers Hardware Reference Manual—Order Number 122704-001

iSBC® 561
SOEMI (Serial OEM Interface) CONTROLLER BOARD

- **Dedicated I/O Controller Provides a Direct Connection of MULTIBUS®-Based Systems to an IBM 4361 Mainframe Host via IBM's SOEMI (Serial OEM Interface) Protocol**
- **Physical Interface is via IBM 3270 Coax with a Maximum Distance of 1.5 km**
- **Maximum Transmission Rate of 2.36 Megabits/Second**
- **Dual I/O Processors Manage Both SOEMI and MULTIBUS® Interfaces**
- **Includes a SMC-to-BNC Cable Assembly to Attach into the IBM 3270 Information Display System**
- **On-Board Diagnostic Capability Provides Operational Status of Board Function and Link with the Host**
- **Supported by a Complete Family of Single Board Computers, Memory, Digital and Analog I/O, Peripheral and Graphics Controllers' Packaging and Software**

The Intel iSBC 561 SOEMI (Serial OEM Interface) Controller Board is a member of Intel's family of single board computers, memory, I/O, peripheral and graphics controller boards. It is a dedicated intelligent I/O controller on a MULTIBUS form-factor printed circuit card. The board allows OEMs of MULTIBUS-based systems a direct, standard link to an IBM System 4361 environment via the SOEMI (Serial OEM Interface). The iSBC 561 Controller also provides 4361 users access to the broad range of applications supported by hundreds of MULTIBUS vendors.

The SOEMI interface is comprised of an IBM System/370 programming interface and a 3270 coax interface. It is a flexible, high speed, point-to-point serial interface offered as a standard feature on the 4361 processor family. The iSBC 561 SOEMI Controller Board contains two processors and provides the necessary intelligence for conversion, control functions, and buffer management between the IBM mainframe and the MULTIBUS system. This board allows an IBM user to distribute control and information to MULTIBUS compatible systems for a variety of applications including factory automation, data acquisition, measurement, control, robotics, process control, communications, local area networking, medical instrumentation, and laboratory automation.



290114-1

*IBM is a trademark of International Business Machine Corp.

October 1986
Order Number: 280114-001

SOEMI INTERFACE OVERVIEW

The Serial OEM Interface (SOEMI) is a new means of connecting Original Equipment Manufacturer (OEM) MULTIBUS-based systems and subsystems to an IBM 4361 mainframe. Previously, the only low-cost way to attach non-IBM equipment into the IBM mainframe environment was to use 3270 emulation software and hardware adaptors. This type of interface is low-speed (approx. 19.6K bits/sec.) and not very flexible as to the type and format of data that can be transferred. The 3270 emulators must mimic the device formats of the displays and printers that are typically attached on this interface; stripping out command characters, carriage return andl line feed characters, etc. The SOEMI Protocol is much faster and more flexible, in that any type of raw data or formatted data may be sent across the connecting coax cable.

The SOEMI attachment into the MULTIBUS system architecture, via the iSBC 561 SOEMI Controller Board, extends the attachment capabilities of the IBM 4361 to a variety of systems, boards, and I/O devices provided by other manufacturers. Figure 1 is an example of the variety achievable on Intel's MULTIBUS (IEEE 796) system architecture.

The SOEMI interface utilizes the System/370 Programming Interface on the IBM 4361 to create the protocols and formats required by a given application for connection to and communication with virtually any type of OEM device.

The System/370 Programming Interface provides the standard System/370 I/O instructions for exchanging data between the host and the MULTIBUS-based system. System/370 applications see MULTIBUS system memory as one or more entities called "spaces." The 4361 host system program writes to and reads from these spaces. The user can define the number of spaces or the layout of fields in the SOEMI interface at his discretion and as required by the application and the MULTIBUS system configuration.

The 3270 coax interface provides the physical connection between the OEM MULTIBUS system and the IBM 4361 host. The coax cable (type RG62AU) can operate over a distance of 1.5 kilometers at a maximum transfer rate of 2.3587 Mbits/second. The distance of 1.5 kilometers can be increased to a maximum of 3 kilometers by installing an IBM 3299 Terminal Multiplexer (repeater) between the IBM 4361 and the MULTIBUS system. The protocol at the coax interface includes a polling mechanism, a set of Write and Read commands, and requires a buffer with an address register at the OEM controller end.

The actual connection to the IBM 4361 is made via the IBM 3270 Information Display System's Display/Printer Adapter (DPA) and/or Work Station Adapter (WSA) coax ports. The DPA can drive up to sixteen 3270/SOEMI coax ports, and is the standard configuration. The WSA is an optional add-on to the IBM 4361 that increases the total number coax ports supported to 40. A typical 4361 configuration can support an aggregate data rate of approximately 45K Bytes/second (approx. 360K bits/second).

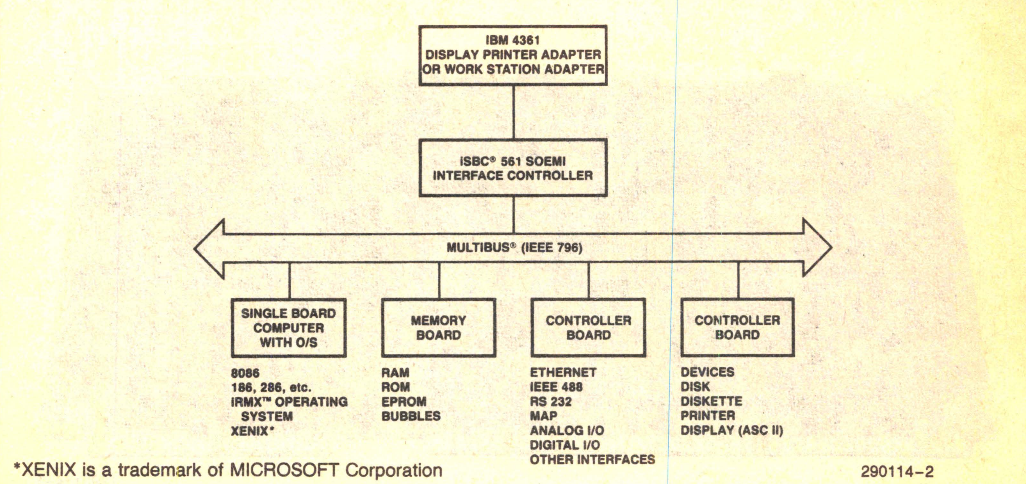


Figure 1. IBM 4361-to-MULTIBUS® Attachment Capability Block Diagram

OPERATING ENVIRONMENT

The iSBC board functions as a slave to the host mainframe, reacting and executing under System/370 program control as as mainframe resource. In addition, it has a full multimaster MULTIBUS interface that allows the board to arbitrate for bus ownership, generate bus clocks, respond to and generate interrupts, etc. With the iSBC 561 controller connected to the 4361 mainframe, all MULTIBUS system resources are available to the IBM host program/controller. From the IBM 4361 side, the mainframe is capable of accessing the entire 16 MBytes of MULTIBUS system memory, 64K Bytes of I/O space, and all on-board resources of the iSBC 561 board. Other intelligent MULTIBUS boards access iSBC 561 controller services through normal interrupt mechanisms.

Using the SOEMI interface in a relatively low-level application may simply require the user to write System/370 application control programs that reside in the IBM 4316 mainframe. A more elaborate implementation would also involve application programs that reside in the MULTIBUS system under its "native" operating environment (i.e., iRMX or XENIX operating systems) and an end-to-end protocol that ties both sets of application programs together.

ARCHITECTURE

The iSBC 561 board is functionally partitioned into three major sections: the front-end section, the common section, and the back-end section (see Figure 2).

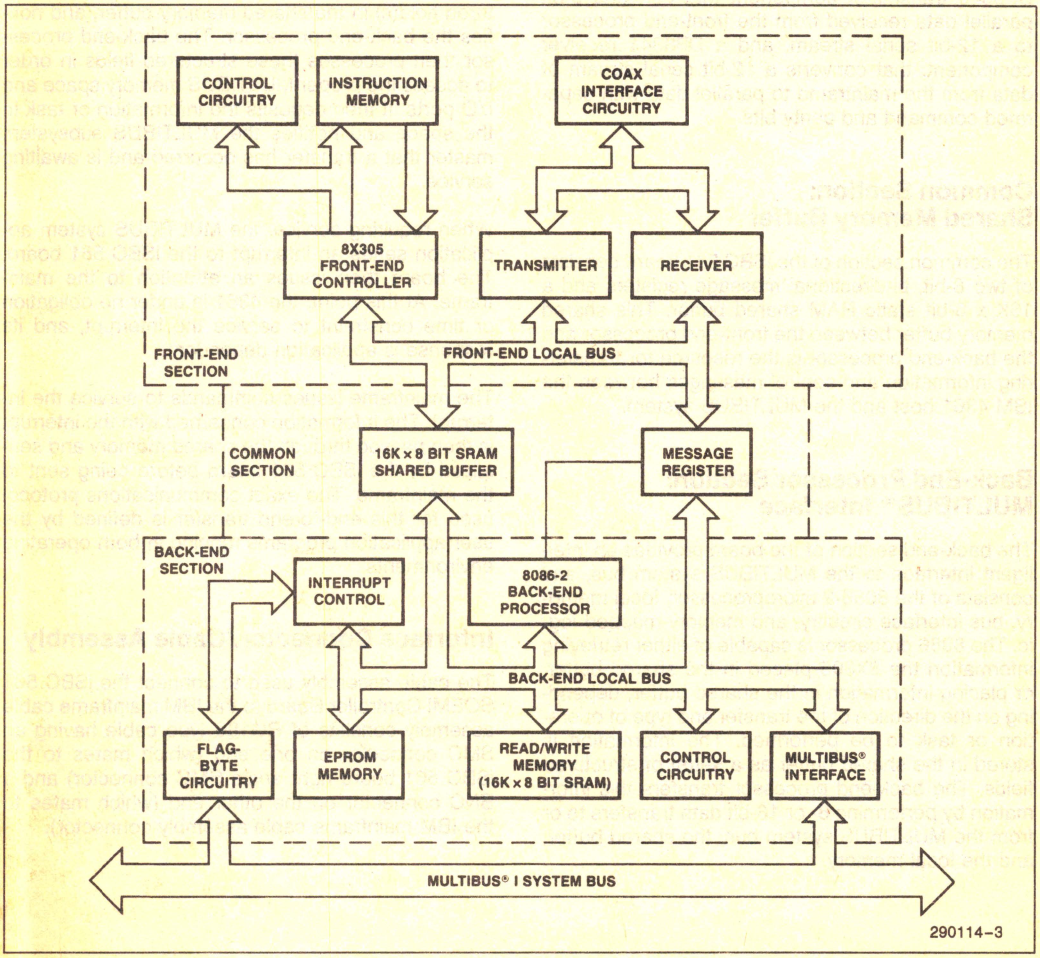


Figure 2. iSBC® 561 SOEMI (Serial OEM Interface) Controller Board Functional Block Diagram

Front-End Processor Section: IBM 4361 Interface

The front-end section of the iSBC 561 Controller board interfaces with the IBM mainframe via the IBM 3270 Information Display System, and consists of an 8X305 Signetics microcontroller, the 8X305 instruction memory, and the coaxial interface. The 8X305 executes the coax commands and places the structured field's instructions in shared memory buffers for subsequent execution by the back-end processor. The front-end instruction memory consists of three 2K x 8-bit PROMs which provide the instruction code for the 8X305 processor and the information needed to generate the various control signals required by the 8X305 to elicit system functions. The information contained in each PROM is not modifiable by the user. The coaxial interface is based on a DP8340 transmitter component that converts 8-bit parallel data received from the front-end processor to a 12-bit serial stream, and a DP8341 receiver component, that converts a 12-bit serial stream of data from the mainframe to parallel data with separated command and parity bits.

Common Section: Shared Memory Buffer

The common section of the iSBC 561 board consists of two 8-bit, bi-directional message registers and a 16K x 8-bit static RAM shared buffer. This shared memory buffer between the front-end processor and the back-end processor is the resource for transferring information and control messages between the IBM 4361 host and the MULTIBUS system.

Back-End Processor Section: MULTIBUS® Interface

The back-end section of the board provides an intelligent interface to the MULTIBUS system bus, and consists of the 8086-2 microprocessor, local memory, bus interface circuitry, and memory-mapped logic. The 8086 processor is capable of either retrieving information the 8X305 placed in the shared buffer, or placing information in the shared buffer, depending on the direction of the transfer and type of operation or task to be performed. The information is stored in the shared buffer as a set(s) of structured fields. The back-end processor transfers this information by performing 8- or 16-bit data transfers to or from the MULTIBUS system bus, the shared buffer, and the local memory.

The control program for this high-speed, back-end processor is resident in two local ROM sites. The processor also has access to 16K bytes of static RAM for local data storage.

The back-end section interfaces to other MULTIBUS boards through two bus controllers, a bus arbiter, and the address, data, and command buffers for access over the 24 address lines and 16 data lines of the MULTIBUS system bus.

OPERATION FLOW

The commands and information passed along the coax by the IBM 4361 host to the iSBC 561 controller represent what is known as a "structured field." The iSBC 561 front-end processor strips out the 12-bit protocol header deposits the remaining structured field(s) in the shared memory buffer, and notifies the back-end processor. The back-end processor then processes these structured fields in order to access the proper MULTIBUS memory space and I/O ports. It then deposits the information or task in the space and notifies the MULTIBUS subsystem master that a transfer has occurred and is awaiting service.

When requiring service, the MULTIBUS system application sends an interrupt to the iSBC 561 board. The board then issues an attention to the mainframe. At this point, the 4361 is under no obligation or time constraint to service the interrupt, and its response is application dependent.

The mainframe issues commands to service the interrupt. The information concerned with the interrupt is then passed through the shared memory and serialized by the iSBC 561 board before being sent to the mainframe. The exact communications protocol used for this end-to-end transfer is defined by the user application programs running in both operating environments.

Interface Connector/Cable Assembly

The cable assembly used to connect the iSBC 561 SOEMI Controller Board to the IBM mainframe cable assembly consists of RG180 type cable having an SMC connector on one end (which mates to the iSBC 561 board right angle SMC connector) and a BNC connector on the other end (which mates to the IBM mainframe cable assembly connector).

SPECIFICATIONS

Operational Characteristics

Back-end processor— Intel 8086-2/5 MHz
— 20-bit address path; 8/16 bit data path

Front-end processor— Signetics 8X305/8 MHz
— 16-bit instruction path; 8-bit data path

Serial Transfer Rate— 2.3587 Mbits/second (max. bit rate)
— 360K bits/second (approx. aggregate throughput)

Serial Transfer Rate— Binary dipulse (with 12-bit serial stream)

Memory Capacity — All iSBC 561 controller board memory is available to on-board firmware only.

Common memory — 16K Bytes of Shared Buffer memory (SRAM @ 0 wait state access)

8086-2 memory — 16K Bytes of EPROM;
— 16K Bytes of SRAM

8X305 memory — 4K Bytes of Instruction memory (EPROM)
— 2K Bytes of Control memory (EPROM)

Physical Characteristics

Width: 30.48 cm (12.00 in)
Height: 17.15 cm (6.75 in)
Depth: 1.78 cm (0.70 in)
Weight: 510 gm (18 oz)

Electrical Characteristics

DC Power Requirements:
Voltage— +5V
Current (Max)—6.28A
Current (Typ)—5.46A
Power Dissipation (Max)—35.5VA

ORDERING INFORMATION

Part Number	Description
iSBC 561	SOEMI (Serial OEM Interface) Controller board

Cable Characteristics

Impedance: coax connector—50 ohms (nominal)
external cable (user furnished)—95 ohms (nominal)

Capacitance: 35 pF/ft

Propagation: 1.6 ns/ft

Environmental Characteristics

Operating Temperature: 0° to 55°C at 200 LFM air velocity

Operating Humidity: 10 to 85% non-condensing (0° to 55°C)

Non-Operating Temperature: −40°C to 75°C

Shock: 30G for a duration of 11 ms with ½ sinewave shape.

Vibration: 0 to 55 Hz with 0.0 to 0.010 inches peak to peak excursion.

Reference Manuals

147048-001— iSBC 561 SOEMI (Serial OEM Interface) Controller Board Hardware Reference Manual (NOT SUPPLIED)

Reference manual may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

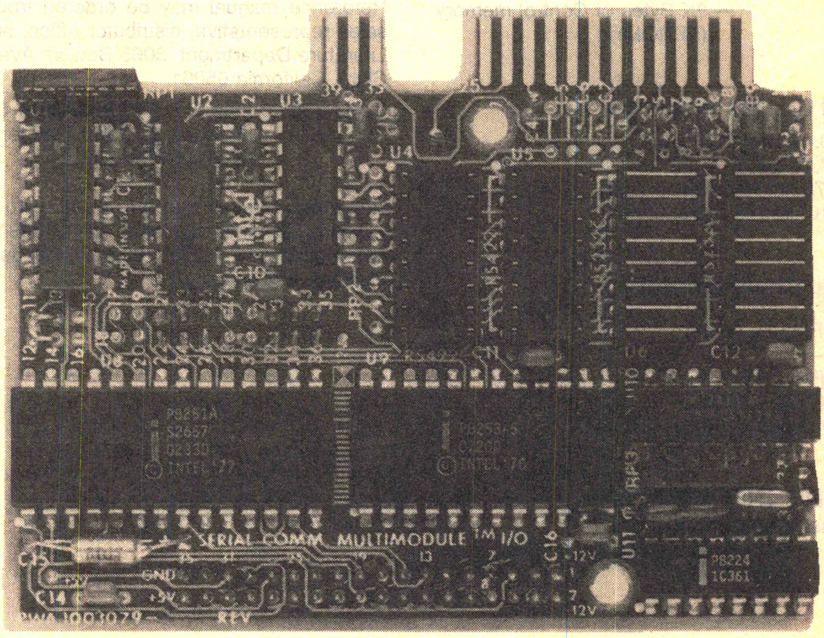
GA33-1585-0 (File No. S370-03—IBM Serial OEM Interface (SOEMI) Reference Manual (NOT SUPPLIED)

Reference manual may be ordered from IBM Advanced Technical Systems; Dept. 3291, 7030-16; Schoenaicherstr. 220; 7030 Boeblingen. Federal Republic of Germany.

iSBX™ 351
SERIAL I/O MULTIMODULE™ BOARD

- iSBX™ Bus Compatible I/O Expansion
- Programmable Synchronous/Asynchronous Communications Channel with RS232C or RS449/422 Interface
- Software Programmable Baud Rate Generator
- Two Programmable 16-Bit BCD or Binary Timer/Event Counters
- Four Jumper Selectable Interrupt Request Sources
- Accessed as I/O Port Locations
- Low Power Requirements
- Single +5V when Configured for RS449/422 Interface
- iSBX Bus On-Board Expansion Eliminates MULTIBUS® System Bus Latency and Increases System Throughput

The Intel iSBX 351 Serial I/O MULTIMODULE board is a member of Intel's new line of iSBX bus compatible MULTIMODULE products. The iSBX MULTIMODULE board plugs directly into any iSBX bus compatible host board offering incremental on-board I/O expansion. The iSBX 351 module provides one RS232C or RS449/422 programmable synchronous/asynchronous communications channel with software selectable baud rates. Two general purpose programmable 16-bit BCD or binary timers/event counters are available to the host board to generate accurate time intervals under software control. The iSBX board is closely coupled to the host board through the iSBX bus, and as such, offers maximum on-board performance and frees MULTIBUS system traffic for other system resources. In addition, incremental power dissipation is minimal requiring only 3.0 watts (assumes RS232C interface).



280236-1

September 1986
Order Number: 280236-001

FUNCTIONAL DESCRIPTION

Communications Interface

The iSBX 351 module uses the Intel 8251A Universal Synchronous/Asynchronous Receiver/Transmitter (USART) providing one programmable communications channel. The USART can be programmed by the system software to individually select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous or asynchronous), data format, control character format, parity, and baud rate are all under program control. The 8251A provides full duplex, double buffered transmit and receive capability. Parity, overrun, and framing error detection are all incorporated in the USART. The command lines, serial data lines, and signal ground lines are brought out to a double edge connector configurable for either an RS232C or RS449/422 interface (see Figure 3). In addition, the iSBX 351 module is jumper configurable for either point-to-point or multidrop network connection.

16-Bit Interval Timers

The iSBX 351 module uses an Intel 8253 Programmable Interval Timer (PIT) providing 3 fully programmable and independent BCD and binary 16-bit interval timers. One timer is available to the system designer to generate baud rates for the USART under software control. Routing for the outputs from the other two counters is jumper selectable to the host board. In utilizing the iSBX 351 module, the systems designer simply configures, via software, each timer independently to meet system requirements. Whenever a given baud rate or time delay is needed, software commands the programmable timers to select the desired function. The functions of the timers are shown in Table 1. The contents of each counter may be read at any time during system operation.

Interrupt Request Lines

Interrupt requests may originate from four sources. Two interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the host board (i.e., receive buffer is full) or a character has been transmitted (i.e., transmit buffer is empty). In addition, two jumper selectable requests can be generated by the programmable timers.

Installation

The iSBX 351 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon hardware to insure the mechanical security of the assembly (see Figures 1 and 2).

Table 1. Programmable Timer Functions

Function	Operation
Interrupt on Terminal Count	When terminal count is reached, an interrupt request is generated. This function is useful for generation of real-time clocks.
Programmable One-Shot	Output goes low upon receipt of an external trigger edge and returns high when terminal count is reached. This function is retriggerable.
Rate Generator	Divide by N counter. The output will go low for one input clock cycle, and the period from one low going pulse to the next is N times the input clock period.
Square-Wave Rate Generator	Output will remain high until one-half the count has been completed, and go low for the other half of the count.
Software Triggered Strobe	Output remains high until software loads count (N). N counts after count is loaded, output goes low for one input clock period.
Hardware Triggered Strobe	Output goes low for one clock period N counts after rising edge counter trigger input. The counter is retriggerable.
Event Counter	On a jumper selectable basis, the clock input becomes an input from the external system. CPU may read the number of events occurring after the counting "window" has been enabled or an interrupt may be generated after N events occur in the system.

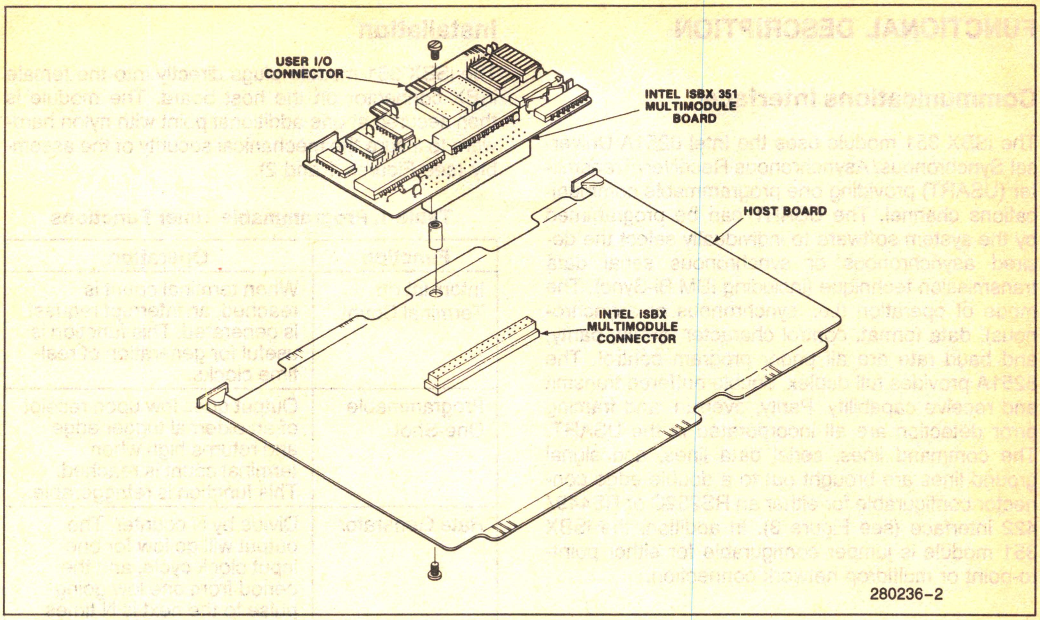


Figure 1. Installation of iSBC® 351 Module on a Host Board

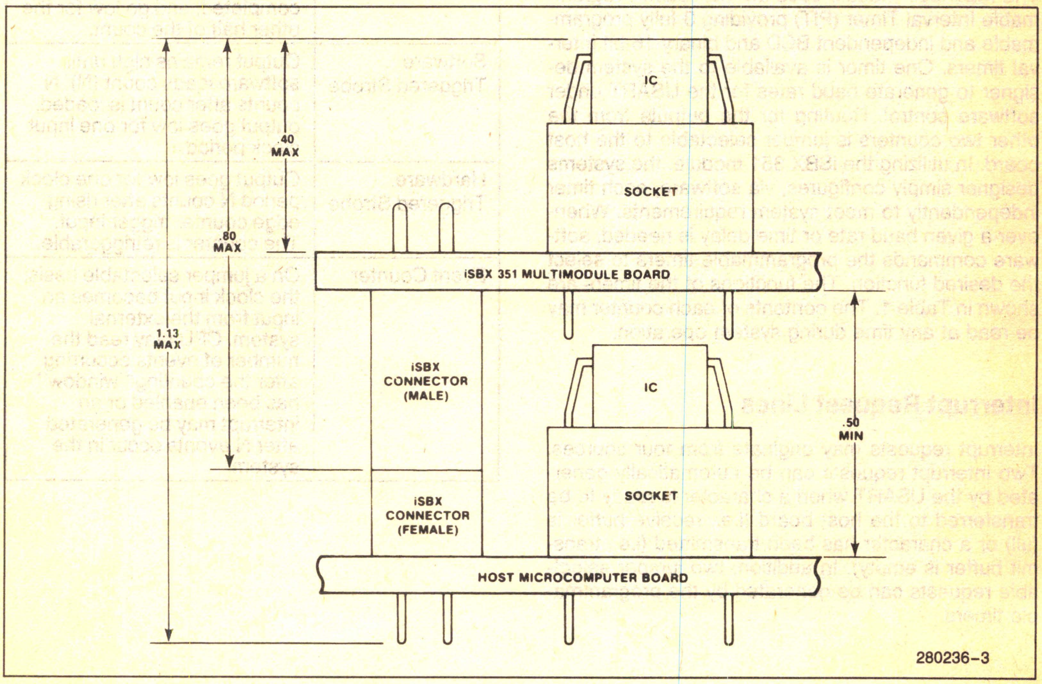


Figure 2. Mounting Clearances (inches)

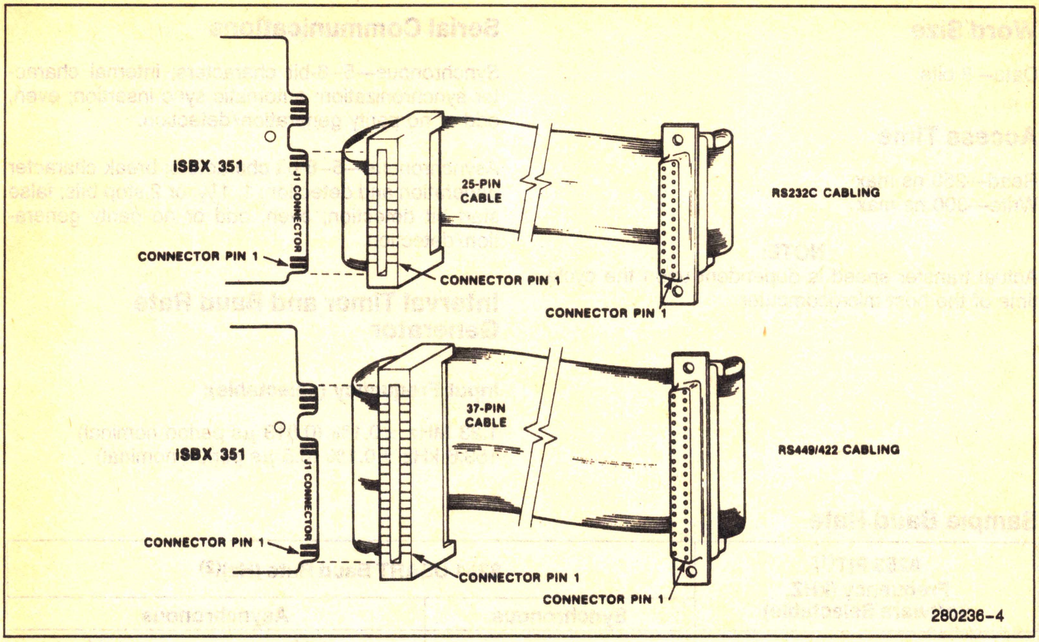


Figure 3. Cable Construction and Installation for RS232C and RS449/422 Interface

SPECIFICATIONS

I/O Addressing

I/O Address for an 8-Bit Host	I/O Address for a 16-Bit Host	Chip Select	Function
X0, X2, X4 or X6	Y0, Y4, Y8 or YC	8251A USART	Write: Data Read: Data
X1, X3, X5 or X7	Y2, Y6, YA or YE	MCS0/ Activated (True)	Write: Mode or Command Read: Status
X8 or XC	Z0 or Z8	8253 PIT	Write: Counter 0 Load: Count (N) Read: Counter 0
X9 or XD	Z2 or ZA	MSC1/Activated (True)	Write: Counter 1 Load: Count N Read: Counter 1
XA or XE	Z4 or ZC		Write: Counter 2 Load: Count (N) Read: Counter 2
XB or XF	Z6 or ZE		Write: Control Read: None

NOTE:
X = The iSBX base address that activates MCS0 & MSC1 for an 8-bit host.
Y = The iSBX base address that activates MCS0 for a 16-bit host.
Z = The iSBX base address that activates MCS1 for a 16-bit host.
The first digit, X, Y or Z, is always a variable, since it will depend on the type of host microcomputer used. Refer to the Hardware Reference Manual for your host microcomputer to determine the first digit of the I/O base address.
The first digit of each port I/O address is listed as "X" since it will change depending on the type of host iSBC microcomputer used. Refer to the Hardware Reference Manual for your host iSBC microcomputer to determine the first digit of the I/O address.

Word Size

Data—8 bits

Access Time

Read—250 ns max
Write—300 ns max

NOTE:
Actual transfer speed is dependent upon the cycle time of the host microcomputer.

Serial Communications

Synchronous—5–8-bit characters; internal character synchronization; automatic sync insertion; even, odd or no parity generation/detection.

Asynchronous—5–8-bit characters; break character generation and detection; 1, 1½, or 2 stop bits; false start bit detection; even, odd or no parity generation/detection.

Interval Timer and Baud Rate Generator

Input Frequency (selectable):

1.23 MHz ±0.1% (0.813 μs period nominal)
153.6 kHz ±0.1% (6.5 μs period nominal)

Sample Baud Rate

8253 PIT(1) Frequency (kHZ, Software Selectable)	8251 USART Baud Rate (Hz)(2)		
	Synchronous	Asynchronous	
		÷16	÷64
307.2	—	19200	4800
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
2.4	2400	150	—
1.76	1760	110	—

NOTES:
1. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.
2. Baud rates shown here are only a sample subset of possible software-programmable rates available. Any frequency from 18.75 Hz to 614.4 kHz may be generated utilizing on-board crystal oscillator and 16-bit Programmable Interval Timer (used here as frequency divider).

Output Frequency

	Rate Generator (Frequency)		Real-Time Interrupt (Interval)	
	Min	Max	Min	Max
Single Timer(1)	18.75 Hz	614.4 kHz	1.63 μs	53.3 ms
Single Timer(2)	2.34 Hz	76.8 kHz	13.0 μs	426.7 ms
Dual Timer(3) (Counters 0 and 1 in Series)	0.000286 Hz	307.2 kHz	3.26 μs	58.25 min
Dual Timer(4) (Counters 0 and 1 in Series)	0.0000358 Hz	38.4 kHz	26.0 μs	7.77 hrs

NOTES:
1. Assuming 1.23 MHz clock input.
2. Assuming 153.6 kHz clock input.
3. Assuming Counter 0 has 1.23 MHz clock input.
4. Assuming Counter 0 has 153.6 kHz clock input.

Interrupts

Interrupt requests may originate from the USART (2) or the programmable timer (2).

Interfaces

iSBX Bus—all signals TTTL compatible.

Serial—configurable of EIA Standards RS232C or RS449/422

EIA Standard RS232C signals provided and supported.

- Clear to Send (CTS)
- Data Set Ready (DSR)
- Data Terminal Ready (DTR)
- Request to Send (RTS)
- Receive Clock (RXC)
- Receive Data (RXD)
- Transmit Clock (DTE TXC)
- Transmit Data (TXD)

EIA Standard RS449/422 signals provided and supported.

- Clear to Send (CS)
- Data Mode (DM)
- Terminal Ready (TR)
- Request to Send (RS)
- Receive Timing (RT)
- Receive Data (RD)
- Terminal Timing (TT)
- Send Data (SD)

Physical Characteristics

Width: 7.24 cm (2.85 inches)
Length: 9.40 cm (3.70 inches)
Height*: 2.04 cm (0.80 inches) iSBX 351 Board
2.86 cm (1.13 inches) iSBX 351 Board and Host Board
Weight: 51 grams (1.79 ounces)
*(See Figure 2)

Serial Interface Connectors

Configuration	Mode(2)	MULTIMODULE™ Edge Connector	Cable	Connector(8)
RS232C	DTE	26-pin(5), 3M-3462-0001	3M(3)-3349/25	25-pin(7), 3M-3482-1000
RS232C	DCE	26-pin(5), 3M-3462-0001	3M(3)-3349/25	25-pin(7), 3M-3483-1000
RS449	DTE	40-pin(6), 3M-3464-0001	3M(4)-3349/37	37-pin(1), 3M-3502-1000
RS449	DCE	40-pin(6), 3M-3464-0001	3M(4)-3349/37	37-pin(1), 3M-3503-1000

NOTES:
1. Cable housing 3M-3485-4000 may be used with the connector.
2. DTE—Data Terminal mode (male connector), DCE—Data Set mode (female connector).
3. Cable is tapered at one end to fit the 3M-3462 connector.
4. Cable is tapered to fit 3M-3464 connector.
5. Pin 26 of the edge connector is not connected to the flat cable.
6. Pins 37, 39, and 40 of the edge connector are not connected to the flat cable.
7. May be used with cable housing 3M-3485-1000.
8. Connectors compatible with those listed may also be used.

Electrical Characteristics

DC Power Requirements

Mode	Voltage	Amps (Max)
RS232C	+5V ±0.25V	460 mA
	+12V ±0.6V	30 mA
	−12V ±0.6V	30 mA
RS449/422	+5V ±0.25V	530 mA

Environmental Characteristics

Temperature: 0°C–55°C, free moving air across the base board and MULTIMODULE board.

Reference Manual

9803190-01— iSBX 351 Serial I/O MULTIMODULE Manual (NOT SUPPLIED)

Reference Manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California, 95051.

ORDERING INFORMATION

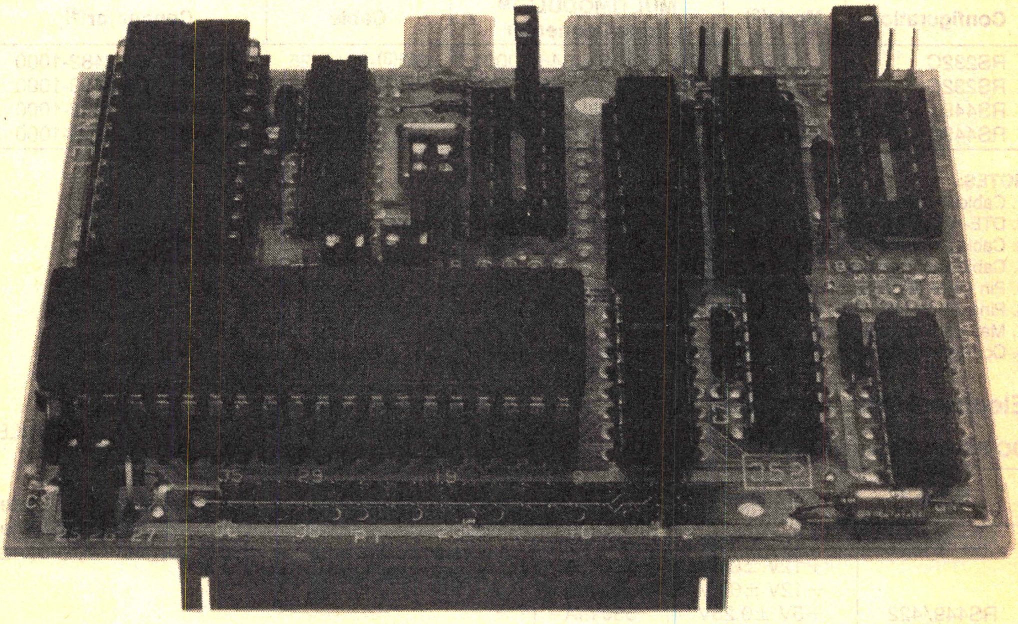
Part Number	Description
SBX 351	Serial I/O MULTIMODULE Board

intel®

iSBX™ 352 BIT SERIAL COMMUNICATIONS MULTIMODULE™ BOARD

- Provides an HDLC/SDLC Half/Full-Duplex Communications Channel for iSBX™ Bus Compatible Microcomputers
- Supports RS232C (Including Modem Support) or RS449/422A Interface
- Single +5V When Configured for RS449/422A Interface
- Software Programmable Baud Rate Generation up to 64K Baud Synchronous and 9.6K Baud Self-Clocking
- Supports Synchronous or Self-Clocking NRZI Point-to-Point, Multidrop and Self-Clocking NRZI SDLC Loop Data Link Interfaces

The Intel iSBX 352 Bit Serial Communications MULTIMODULE board offers incremental on-board I/O expansion support for ISO/CCITT's HDLC or IBM's SDLC communication. Plugging directly into any iSBX bus compatible host board, the iSBX 352 module provides one RS232C or RS449/422A programmable bit serial communications channel with software selectable baud rates (up to 64K baud for half-duplex synchronous operations). Data link interfaces supported are: synchronous point-to-point, multidrop and SDLC loop. The phase lock loop feature provides NRZI self-clocking 9.6K baud operation.



210218–1

October 1986
Order Number: 210218-001

FUNCTIONAL DESCRIPTION

Communications Interface

The iSBX 352 module uses the Intel 8273 Programmable HDLC/SDLC Protocol Controller. The iSBX 352 module provides one bit-serial communications channel for iSBX bus compatible host microcomputers. (See Figure 1.) An iSBC microcomputer or MULTIBUS-based application is easily connected to an HDLC/SDLC point-to-point, multidrop, or an SDLC loop configuration.

The High-Level Data Link Control (HDLC) is the International Standards Organization (ISO) standard discipline used to implement X.25 packet switching communications. The Synchronous Data Link Control (SDLC) is an IBM communication protocol used to implement the System Network Architecture (SNA). Both protocols, HDLC and SDLC, are bit oriented, code independent, and suport full-duplex operations.

Data Link Interface

The control lines, serial data lines and signal ground lines are brought out to the double edge connector of the iSBX 352 module and are configurable for RS232C or RS449/422A interface (see Figure 2).

Addressing an iSBX 352 board by using a port address, the program performs the 8-bit data transfer required, using buffered or non-buffered transmit/receive and abort sequences.

Serial data transfer control is provided by the 8273 controller of the iSBX 352 module which interfaces the parallel iSBX bus to the serial channel. During a transmit sequence, the iSBX 352 module accepts data and commands from the iSBX bus interface, translates and formats the data into HDLC/SDLC protocol formats, provides the proper RS232C or RS422A interface control signals, and passes data onto the serial channel. The receive operation is the inverse of the previous sequence.

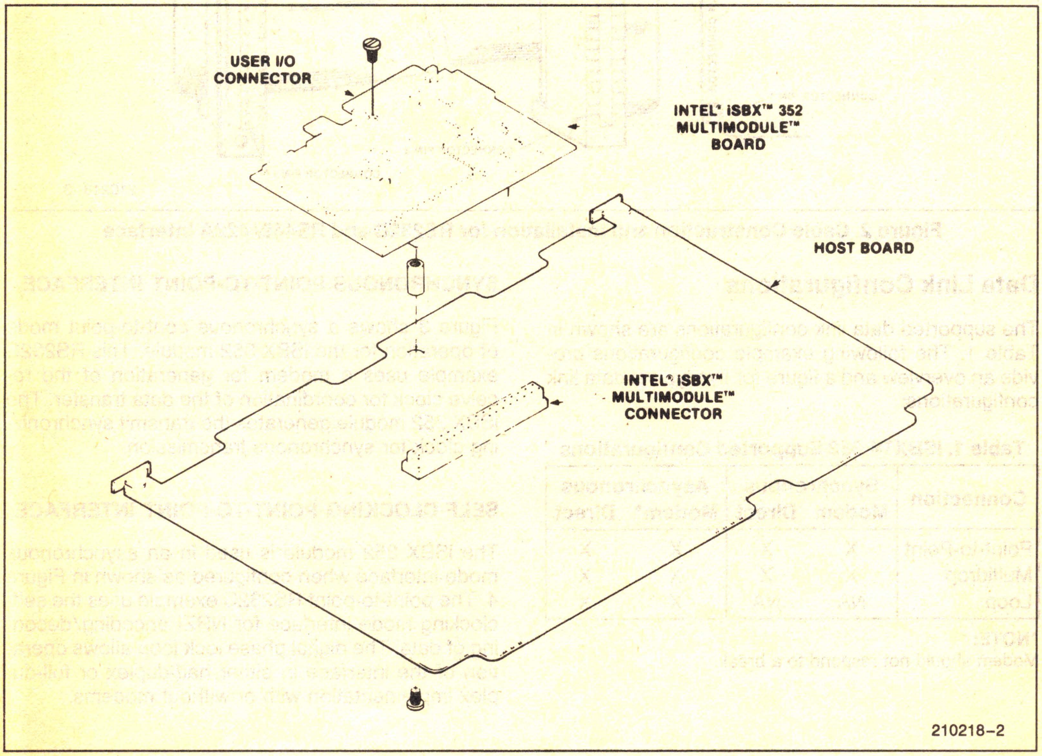


Figure 1. Installation of iSBX™ 352 MULTIMODULE Board on a Host Board

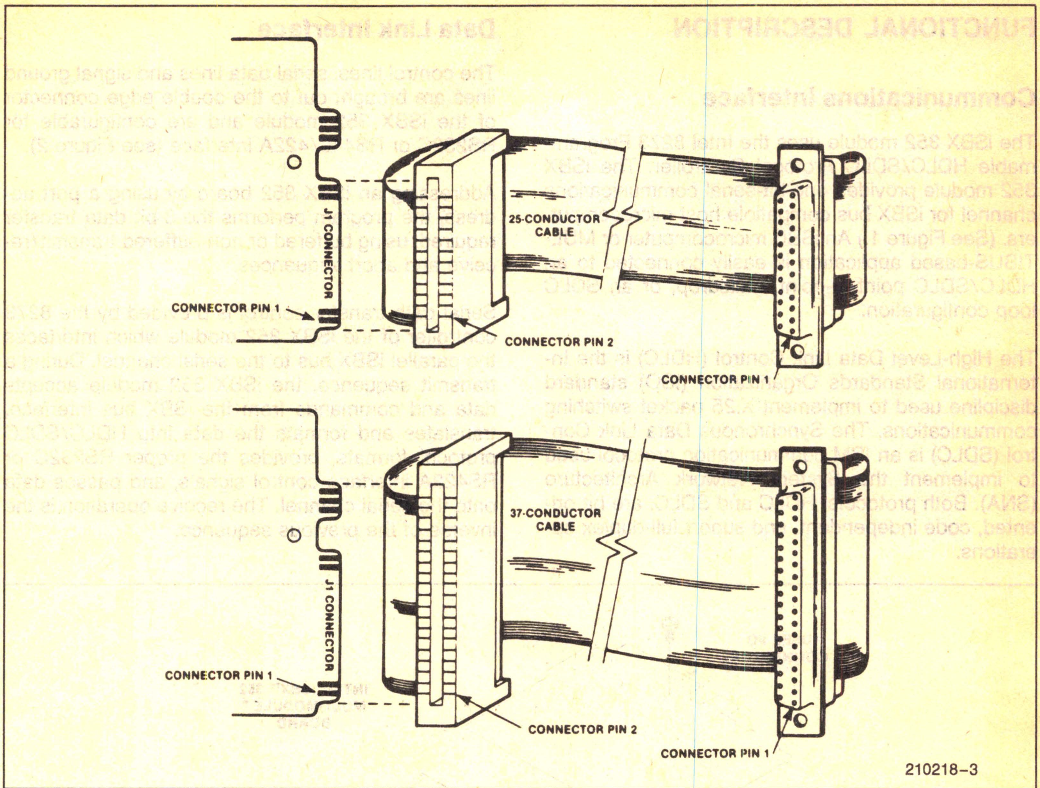


Figure 2. Cable Construction and Installation for RS232C and RS449/422A Interface

Data Link Configurations

The supported data link configurations are shown in Table 1. The following example configurations provide an overview and a figure for five typical data link configurations:

Table 1. iSBX™ 352 Supported Configurations

Connection	Synchronous Modem	Synchronous Direct	Asynchronous Modem*	Asynchronous Direct
Point-to-Point	X	X	X	X
Multidrop	X	X	X	X
Loop	NA	NA	X	X

***NOTE:**
Modem should not respond to a break.

SYNCHRONOUS POINT-TO-POINT INTERFACE

Figure 3 shows a synchronous pont-to-point mode of operation for the iSBX 352 module. This RS232C example uses a modem for generation of the receive clock for coordination of the data transfer. The iSBX 352 module generates the transmit synchronizing clock for synchronous transmission.

SELF-CLOCKING POINT-TO-POINT INTERFACE

The iSBX 352 module is used in an asynchronous mode interface when configured as shown in Figure 4. The point-to-point RS232C example uses the self-clocking mode interface for NRZI encoding/decoding of data. The digital phase lock loop allows operation of the interface in either half-duplex or full-duplex implementation with or without modems.

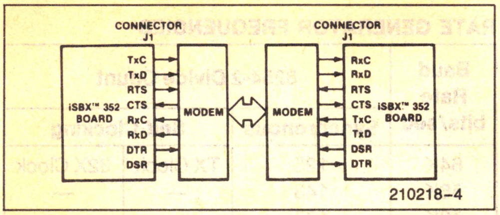


Figure 3. Synchronous Point-to-Point Modem Interface Configuration Example—RS232C

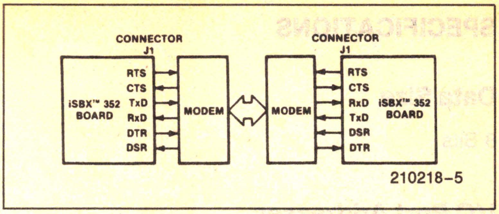


Figure 4. Self-Clocking Point-to-Point Modem Interface Configuration Example—RS232C

SYNCHRONOUS MULTIDROP

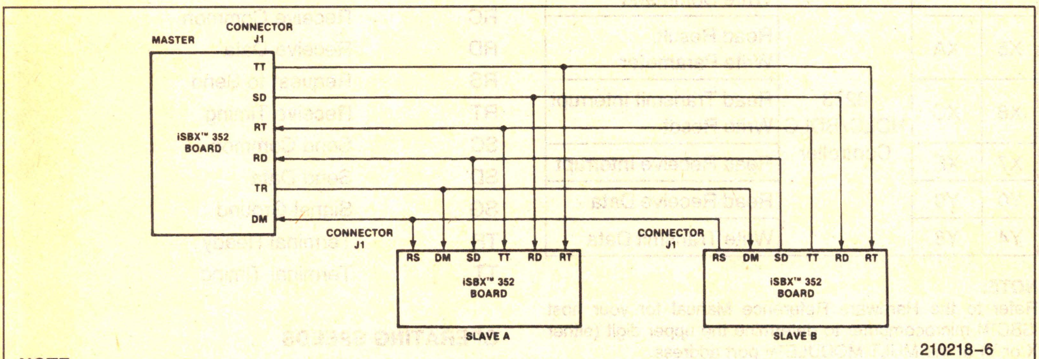
The iSBX 352 MULTIMODULE is used in both a master and a slave mode in the RS449/422A example shown in Figure 5. This synchronous multidrop application is effective for high-speed data transfers between slave stations and a central master station.

ASYNCHRONOUS SELF-CLOCKING MULTIDROP

The iSBX 352 MULTIMODULE example in Figure 6 shows a master and multiple slaves in a multidrop configuration. This self-clocking example uses the 8273 digital phase lock loop and NRZI data encoding.

SDLC Loop

The SDLC self-clocking loop configuration shown in Figure 7 permits longer networks since each secondary slave station is a repeater set in one-bit-delay mode. The data sent out by the primary station (the loop controller) are relayed bit-for-bit through each secondary station and finally back to the master station.



NOTE:
The last slave device in the system must contain termination resistors on all signal lines received by the slave board. The master device must contain resistors on all received signal lines.

Figure 5. Synchronous Multidrop Network Configuration Example—RS422A

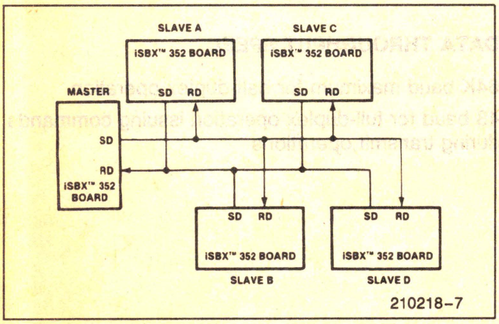


Figure 6. Self-Clocking Multidrop Configuration Example—RS422A

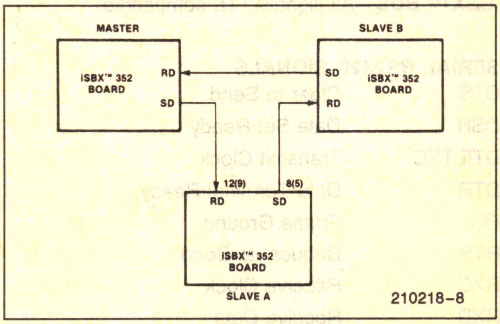


Figure 7. Self-Clocking SDLC Loop Network Configuration Example

SPECIFICATIONS

Data Size

8 Bits

I/O Port Addresses

Port Address 8-Bit	Port Address 16-Bit	Device Selected	Function Performed
X0	X0	8254-2 PIT	Read Counter 0 Write Counter 0
X1	X2	8254-2 PIT	Read Counter 1 Write Counter 1
X2	X4	8254-2 PIT	Read Counter 2 Write Counter 2
X3	X6	8254-2 PIT	Write Control
X4	X8	8273 HDLC/SDLC Controller	Read Status Write Command
X5	XA	8273 HDLC/SDLC Controller	Read Result Write Parameter
X6	XC	8273 HDLC/SDLC Controller	Read Transmit Interrupt Write Reset
X7	XE	8273 HDLC/SDLC Controller	Read Receive Interrupt
Y0	Y0	8273 HDLC/SDLC Controller	Read Receive Data
Y4	Y8	8273 HDLC/SDLC Controller	Write Transmit Data

NOTE:
Refer to the Hardware Reference Manual for your host iSBC™ microcomputer to determine the upper digit (either X or Y) of the MULTIMODULE™ port address.

Interfaces

iSBX™ BUS—All signals TTL compatible

SERIAL RS232C SIGNALS

CTS	Clear to Send
DSR	Data Set Ready
DTE TXC	Transmit Clock
DTR	Data Terminal Ready
FG	Frame Ground
RTS	Request to Send
RXC	Receive Clock
RXD	Receive Data
SG	Signal Ground
TXD	Transmit Data

RATE GENERATOR FREQUENCIES

Baud Rate bits/sec	8254-2 Divide Count		
	Synchronous	Self-Clocking	
		TX Clock	32X Clock
64K	125	—	—
56K	143	—	—
48K	167	—	—
19.2K	417	—	—
9.6K	833	833	26
4.8K	1,667	1,667	52
2.4K	3,333	3,333	104
1.2K	6,667	6,667	208
0.6K	13,333	13,333	417
0.3K	26,667	26,667	833

NOTE:
All numbers are in decimal notation.

SERIAL RS449/422A SIGNALS

CS	Clear to Send
DM	Data Mode
RC	Receive Common
RD	Receive Data
RS	Request to Send
RT	Receive Timing
SC	Send Common
SD	Send Data
SG	Signal Ground
TR	Terminal Ready
TT	Terminal Timing

OPERATING SPEEDS

24 MHz on-board crystal

8 MHz clocking of the 8254-2 PIT

4 MHz clocking of the 8273 Device

DATA THROUGHPUT SPEED

64K baud maximum for half-duplex operation

48 baud for full-duplex operation issuing commands during transmit operations

SERIAL INTERFACE CONNECTORS

Configuration	Mode(2)	MULTIMODULE™ Edge Connector	Cable	Connector
RS232C	DTE	26-pin(5), 3M-3462-0001	3M(3)-3349/25	25-pin(7), 3M-3482-1000
RS232C	DCE	26-pin(5), 3M-3462-0001	3M(3)-3349/25	25-pin(7), 3M-3483-1000
RS449	DTE	40-pin(6), 3M-3464-0001	3M(4)-3349/37	37-pin(1), 3M-3502-1000
RS449	DCE	40-pin(6), 3M-3464-0001	3M(4)-3349/37	37-pin(1), 3M-3503-1000

NOTES:
1. Cable housing 3M-3485-4000 may be used with the connector.
2. DTE-Data Terminal Equipment mode (male connector); DCE—Data Set Equipment mode (female connector).
3. Cable is tapered at one end to fit the 3M-3462 connector.
4. Cable is tapered to fit 3M-3464 connector.
5. Pin 26 of the edge connector is not connected to the flat cable.
6. Pins 38, 39, and 40 of the edge connector are not connected to the flat cable.
7. May be used with the cable housing 3M-3485-1000.

ELECTRICAL CHARACTERISTICS

DC POWER REQUIREMENTS

Interface	Voltage	Current (max)	Total Power
RS232C	+5 ±0.25V −12 ±0.6V +12 ±0.6V	595 mA 30 mA 30 mA	3.8 watts
RS449/422A	+5 ±0.25V	775 mA	4.1 watts

Environmental Characteristics

Temperature— 0°C–55°C, free moving air across base board and MULTIMODULE board

Humidity — to 90%, without condensation

Physical Characteristics

Width: 727 cm (2.85 inches)

Length: 9.40 cm (3.70 inches)

Height: 1.40 cm (0.56 inches)

Weight: 72 gm (2.53 ounces)

Reference Manual (Not Supplied)

143983: iSBX 352 Bit Serial Communications MULTIMODULE Board Hardware Reference Manual.

Reference manuals may be ordered from any Intel sales representative, distributor office or from Intel Literature Department, 3065 Bowers Ave., Santa Clara, California 95051.

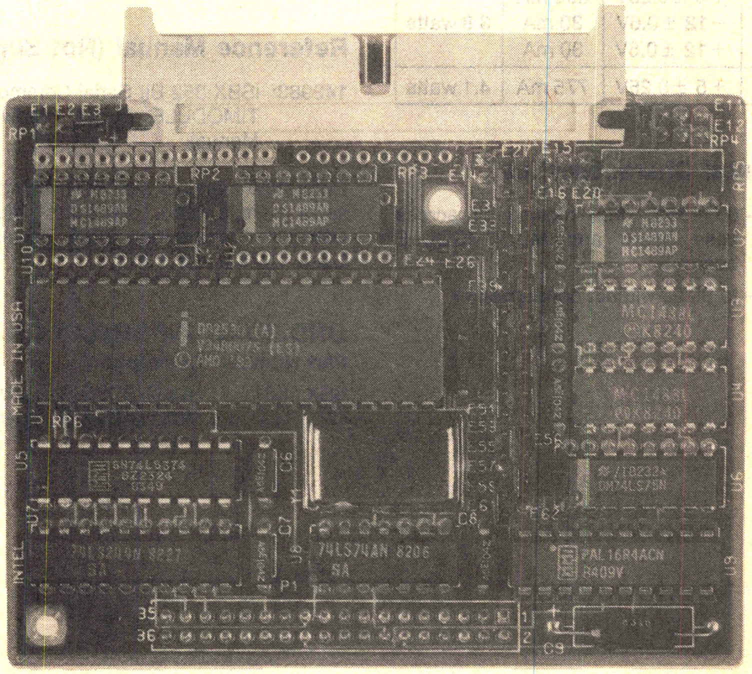
ORDERING INFORMATION

Part Number	Description
SBX 352	HDLC/SDLC Serial I/O MULTIMODULE Board

iSBX™ 354 DUAL CHANNEL SERIAL I/O MULTIMODULE™ BOARD

- Two RS232C or RS422A/449 Programmable Synchronous/ Asynchronous Communications Channels
- Programmable Baud Rate Generation for Each Channel
- Full Duplex Operation
- iSBX™ Bus Compatible I/O Expansion
- Supports HDLC/SDLC, NRZ, NRZI or FM Encoding/Decoding
- Three Interrupt Options for Each Channel
- Low Power Requirements

The Intel iSBX 354 Serial I/O MULTIMODULE board is a member of Intel's line of iSBX compatible MULTIMODULE products. The iSBX MULTIMODULE board plugs directly into any iSBX bus compatible host board offering incremental on-board I/O expansion. Utilizing Intel's 82530 Serial Communications Controller component, the iSBX 354 module provides two RS232C or RS422A/449 programmable synchronous/asynchronous communications channels. The 82530 component provides two independent full duplex serial channels, on chip crystal oscillator, baud-rate generator and digital phase locked loop capability for each channel. The iSBX board connects to the host board through the iSBX bus. This offers maximum on-board performance and frees the MULTIBUS® System bus for use by other system resources.



280045–1

October 1986
Order Number: 280045-002

FUNCTIONAL DESCRIPTION

Communications Interface

The iSBX 354 module uses the Intel 82530 Serial Communications Controller (SCC) component providing two independent full duplex serial channels. The 82530 is a multi-protocol data communications peripheral designed to interface high speed communications lines using Asynchronous, Byte-Synchronous and Bit-Synchronous protocols to Intel's microprocessor based board and system level products. The mode of operation (i.e. asynchronous or synchronous), data format, control character format, and baud-rate generation are all under program control. The 82530 SCC component can generate and check CRC codes in any Synchronous mode and can be programmed to check data integrity in various modes. The command lines, serial data lines, and signal ground lines are brought out to a double edge connector.

The iSBX 354 module provides a low cost means to add two serial channels to iSBC® boards with 8 or 16 bit MULTIMODULE interfaces. In the factory default configuration, the iSBX 354 module will support two RS232C interfaces. With user supplied drivers and termination resistors, the iSBX 354 module can be reconfigured to support RS422A/449 communication interfaces with support on Channel A only for multidrop control from the base board. Both channels can be configured as DTE or DCE with RS232C interfaces.

Interrupt Request Line

The 82530 SCC component provides one interrupt to the MINTRO signal of the iSBX interface. There are six sources of interrupts in the SCC component (Transmit, Receive and External/Status interrupts in both channels). Each type of interrupt is enabled under program control with Channel A having higher priority than Channel B, and with Receive, Transmit

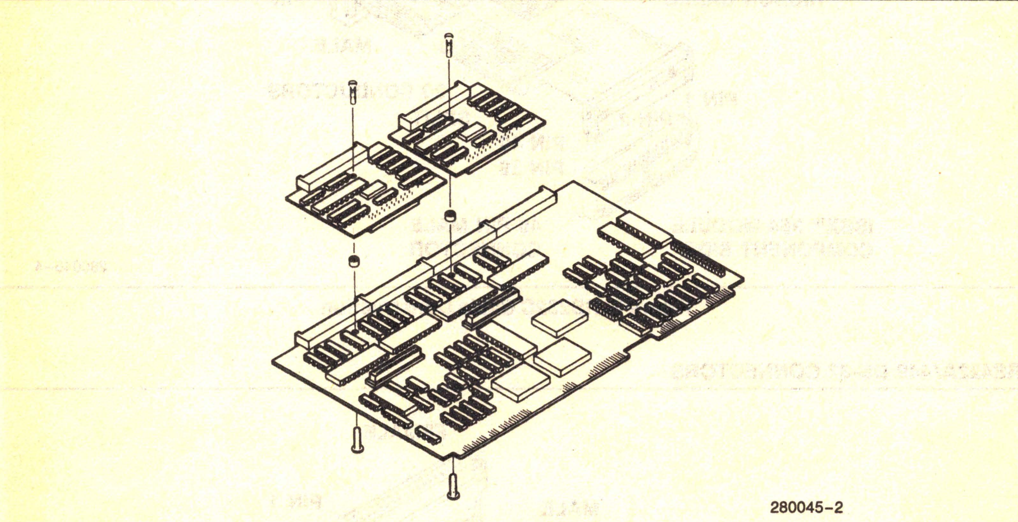


Figure 1. Installation of 2 iSBX™ 354 MULTIMODULE™ Boards on an iSBC® Board

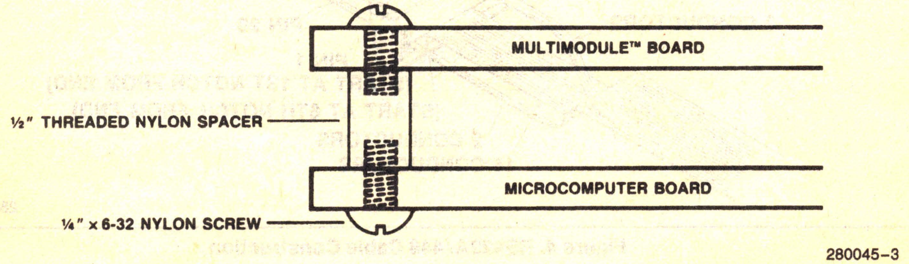


Figure 2. Mounting Technique

and External/Status interrupts prioritized in that order within each channel.

Installation

The iSBX 354 module plugs directly into the female iSBX connector on the host board. The module is then secured at one additional point with nylon hardware to insure the mechanical security of the assembly. Figures 1 and 2 demonstrate the installation of the iSBX 354 MULTIMODULE board on a Host Board. Figures 3 and 4 provide cabling diagrams.

Programming Considerations

The Intel 82530 SCC component contains several registers that must be programmed to initialize and control the two channels. Intel's iSBX 354 Module Hardware Reference Manual (Order #146531-001) describes these registers in detail.

RS232C DB-25 CONNECTORS

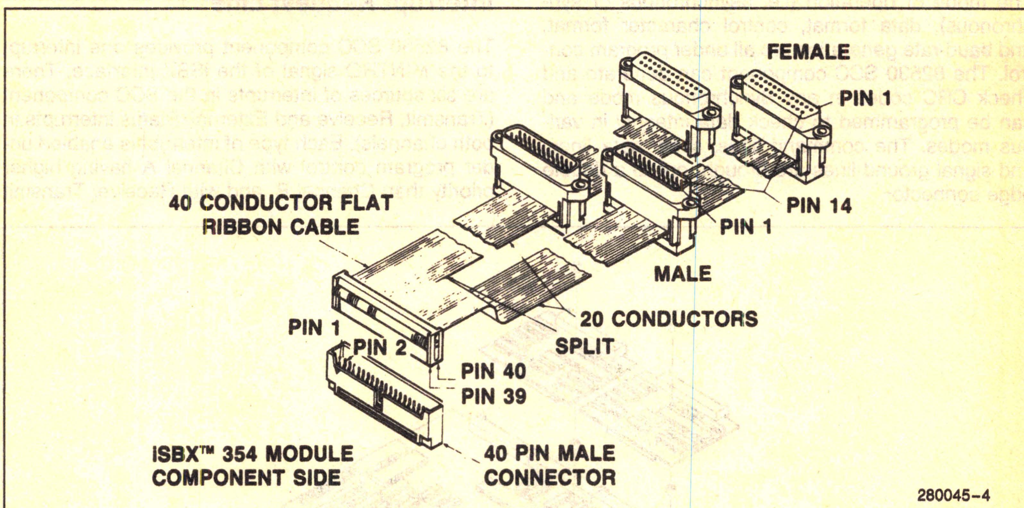


Figure 3. RS232C Cable Construction

RS422A/449 DB-37 CONNECTORS

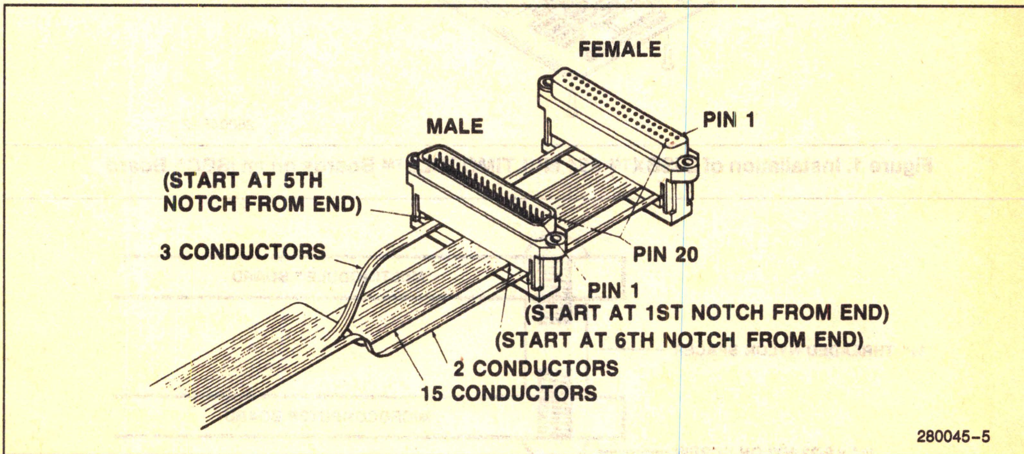


Figure 4. RS422A/449 Cable Construction

SPECIFICATIONS

Word Size

Data—8 bits

Clock Frequency

4.9152 MHz

Serial Communications

Synchronous—Internal or external character synchronization on one or two synchronous characters

Asynchronous—5–8 bits and 1, 1½ or 2 stop bits per character; programmable clock factor; break detection and generation; parity, overrun, and framing error detection

Sample Baud Rate:

Synchronous X1 Clock	
Baud Rate	**82530 Count Value (Decimal)**
64000	36
48000	49
19200	126
9600	254
4800	510
2400	1022
1800	1363
1200	2046
300	8190
Asynchronous X16 Clock	
Baud Rate	**82530 Count Value (Decimal)**
19200	6
9600	14
4800	30
2400	62
1800	83
1200	126
300	510
110	1394

INTERFACES

iSBX™ Bus: Meets the iSBX Specification, Compliance Level: D8 I

Serial: Meets the EIA RS232C standard on Channels A and B. Meets the EIA RS422A/449 standard on Channels A and B, Multi-drop capability on Channel A only.

Signals Provided

RS232C DTE

-Transmit Data
-Receive Data
-Request to Send
-Clear to Send
-Data Set Ready
-Signal Ground
-Carrier Detect
-Transmit Clock (2)
-Receive Clock
-Data Terminal Ready
-Ring Indicator

RS232C DCE

-Transmit Data
-Receive Data
-Clear to Send
-Data Set Ready
-Signal Ground
-Carrier Detect
-Transmit Clock (2)
-Receive Clock
-Ring Indicator

RS422A/449

-Send Data
-Receive Timing
-Receive Data
-Terminal Timing
-Receive Common

I/O Port Addresses

Port Address 8-Bit 16-Bit	Function
X0	Read Status Channel B Write Command Channel B
X2	Read Data Channel B Write Data Channel B
X4	Read Status Channel A Write Command Channel A
X6	Read Data Channel A Write Data Channel A
Y0	Read Disable RS422A/449 Buffer Write Enable RS422A/449 Buffer

NOTES:
1. The "X" and "Y" values depend on the address of the iSBX interface as viewed by the base board.
2. "X" corresponds with Activation of the MCS0/interface signal; "Y" corresponds with Activation of the MCS1/interface signal.

Power Requirements

+5V at 0.5A
+12V at 50 mA
−12V at 50 mA

Physical Characteristics

Width: 2.85 inches
Length: 3.70 inches
Height: 0.8 inches
Weight: 85 grams

ENVIRONMENTAL CHARACTERISTICS

Temperature: 0°C to 55°C operating at 200 linear feet per minute across baseboard and MULTIMODULE board

Humidity: To 90%, without condensation

ORDERING INFORMATION

Part Number	Description
iSBX 354	Dual Channel I/O MULTIMODULE

REFERENCE MANUAL

146531-001—iSBX 354 Channel Serial I/O Board Hardware Reference Manual

Reference manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051.

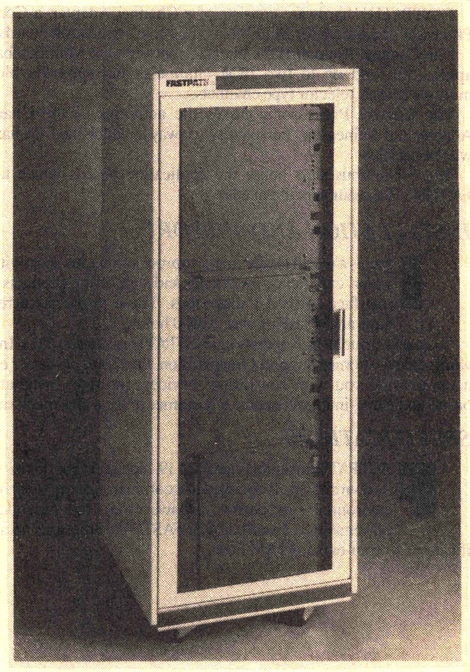
FASTPATH™

Intel's Interconnect Solution

FASTPATH—THE OEM PLATFORM FOR THE DISTRIBUTED PROCESSING ENVIRONMENT

Until now there hasn't been a simple way to get high performance direct access to an IBM mainframe. Now Intel has changed that with FASTPATH.

Now you can add the power of an IBM mainframe to a local area network or to a department minicomputer, or even connect specialized peripherals to a mainframe.



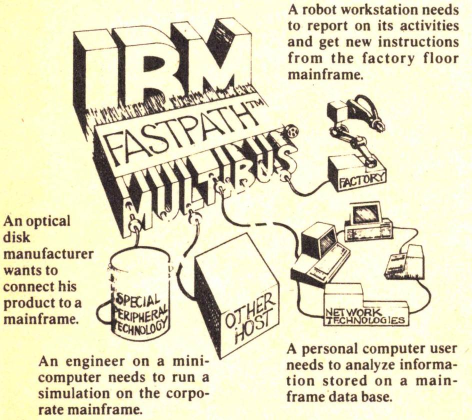
FASTPATH

FASTPATH is a Channel to Multibus Adapter connecting the Intel MULTIBUS® architecture to an IBM System 370 I/O channel and other processors conforming to the 370 I/O specification.

FASTPATH acts as a high-speed interface, between the channel and a Multibus-based application, achieving throughput rates of up to 3 million bytes per second.

What makes this possible is the marriage of 14 years of Intel experience and technology in interfacing to IBM mainframes with the power and flexibility of Multibus.

Multibus is an industry standard, IEEE 796, for interconnecting microprocessor-based boards. It's such a popular standard that over 200 manufacturers offer over 1350 different Multibus-based modules—LAN controllers, communications controllers, CPUs, memories, etc. This broad range of products allows you to choose the exact elements needed for a custom solution to your application needs.



The common thread is cooperative processing, the need to interact with a mainframe, to share resources with it.

APPLICATIONS

Some typical FASTPATH applications include connecting an IBM mainframe to one or more • Local Area Networks • Wide Area Networks • dissimilar hosts • special peripherals—all running your choice of protocol.

FEATURES

Data Streaming transfer mode, at 3 Megabytes per second, is supported by FASTPATH on an IBM System 370 I/O interface as well as up to 2 Megabytes per second in high-speed data transfer mode.

An optional Two Channel Switch connects FASTPATH to two channels on one IBM host or one channel on each of two hosts (field installable).

A Speed Matching Buffer optimizes channel throughput by providing flowthrough buffering between the channel and slower applications hardware.

An optional remote maintenance feature provides for the remote

diagnosis of hardware problems.

FASTPATH's hardware consists of a Channel Adapter, a Control Processor and 6 slots for Multibus-compatible application boards.

The Channel Adapter is the interface between the Multibus-based application and up to two IBM channels, and supports block-multiplexer and selector type channels.

The Control Processor controls the activities of the Channel Adapter and defines the "personality" or way in which the I/O channel will be handled.

The 6 Multibus slots house the application boards which tailor FASTPATH to your specific needs.

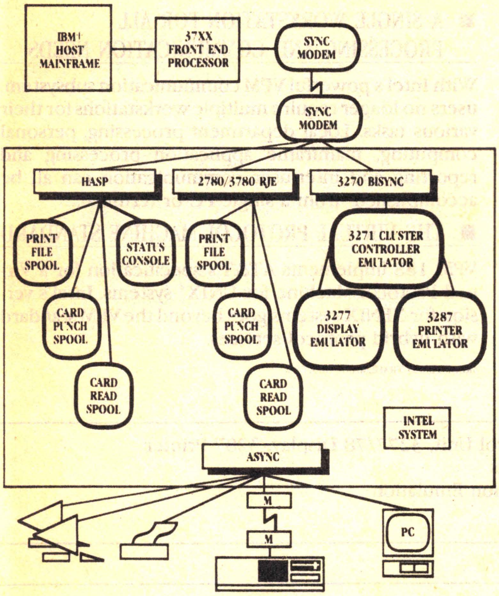
INSTALLATION AND SUPPORT

Intel is a world leader in microprocessor technology with 14 years' experience producing and supporting products that interface to IBM mainframes. These products currently have an MTBF of over 10,000 hours.

To ensure trouble-free operation FASTPATH is supported by Intel's worldwide Customer Support Organization. Over 900 customer engineers provide you with consulting services, training, installation, onsite and carry-in maintenance, a customer hotline and much more.

SPECIFICATIONS

FASTPATH fits into a standard 19-inch rack and comes with its own chassis. It operates in a commercial computer room environment and derives its power from a standard facility power source. Installation of FASTPATH requires no modifications to your existing IBM host.



VPM 188 ASYNC/BISYNC COMMUNICATION SUBSYSTEM

- ***Virtual Protocol Machine delivers asynchronous and bisynchronous lines and protocols for Intel XENIX* systems and networks***
- ***Mainframe link for host data, application, and report access provided by IBM protocol emulation of HASP, 2780/3780 RJE and 3270 bisync protocols***
- ***Async communications support for terminal multiplexing, serial printers, async modems and serial system-to-system links***
- ***Single hybrid subsystem for very cost effective and flexible multiple service communication support***
- ***Full menu-driven installation, administration and user interface for ease of use***

VPM 188 ASYNC/BISYNC COMMUNICATION SERVICES

The VPM 188 Async/Bisync Communication Subsystem delivers a wide variety of communication services. IBM mainframe access is provided via emulation of IBM bisync network protocols and devices:

- 3271 Model 2 Cluster Controller, 3277/78 Displays, and 3287 Printers for interactive host access
- 2780/3780 Remote Job Entry (RJE) Workstation for batch host access
- Multileaving HASP RJE Workstation for batch host access

Comprehensive asynchronous communication support is provided, including:

- Async terminal multiplexing either directly attached or remotely connected via dial-up async modems
- Serial printer control
- System-to-system link support through serial async line using UUCP; either directly attached or remotely connected via async modem

FLEXIBLE MULTILINE CONTROL AND CONFIGURATION

The subsystem controls 8, 10 or 12 communication lines in a wide set of user selected configurations. It allows dynamic selection of line types at install and boot time; the number of bisync lines and protocols and the number of async lines can be configured for specific application requirements. Both interactive and batch mainframe access is supported via emulation of IBM's most popular network and device protocols.

MENU-DRIVEN INTERFACE FOR INTEGRATION AND EASE OF USE

The VPM subsystem's flexibility and power is delivered to users and administrators through a comprehensive menu system. The menus lead users and administrators through installation, generation, administration, and use of the subsystem in a nonconfrontive, easy to use manner. Rapid productivity gains results.

JUNE, 1986
ORDER NUMBER: 270216-001

■ ADVANCED SUBSYSTEM FEATURES

Support for simultaneous operation of multiple mixed line types and bisync protocol emulations is enabled using VPM's dynamic line configuration and protocol downloading features.

OpenNET™ compatible network operation allows bisync emulation services and async communication services to be accessed by remote XENIX and DOS users across OpenNET for cost-effective gateway operation. May require OpenNET Virtual Terminal.

Screen and print data can be moved to any system or network file for subsequent processing by standard XENIX and/or DOS applications. Addition of custom applications can enable IBM compatible distributed DP.

■ A SINGLE WORKSTATION FOR ALL PROCESSING AND COMMUNICATION NEEDS

With Intel's powerful VPM communication subsystem, users no longer require multiple workstations for their various tasks. Local department processing, personal computing, mainframe application processing and reporting and inter-user communication can all be accomplished from a single PC or terminal.

■ THE VIRTUAL PROTOCOL MACHINE STANDARD

VPM 188 implements AT&T's specification for a Virtual Protocol Machine for UNIX* systems. Intel's version for XENIX systems goes beyond the VPM standard with hybrid protocol services.

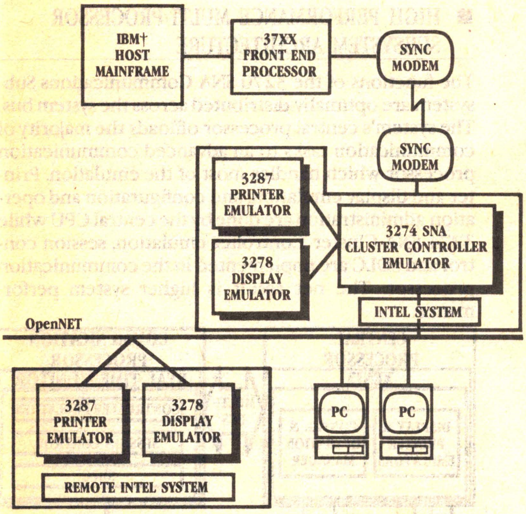
*REGISTERED TRADEMARK OF AT&T

TECHNICAL SPECIFICATIONS

3270 Bisync Emulation — 3271 Model 2 Cluster Control Unit, 3277/78 Display, 3287 Printer
HASP Emulation — Multileaving HASP RJE Workstations
2780/3780 RJE Emulation — 2780/3780 RJE Workstation Emulation
9600 bps full and half-duplex line support

ORDERING INFORMATION

VPM188DK	The base Virtual Protocol Machine for 188/48 and 188/56 controllers in Intel XENIX systems. Controls 8, 10 or 12 RS232 lines to be async and/or bisync. All async support included. Package includes software and documentation. Prerequisite is XENIX system with iBASE.
HASP188DK	Multileaving HASP RJE workstation emulator. Supports both transparent and non-transparent mode HASP protocols. Package includes HASP emulation software, installation instructions, user guide and administrator guide. Prerequisite is VPM188DK.
RBTE188DK	2780/3780 RJE workstation emulator for Remote Batch Terminal Emulation (RBTE) across bisync lines/networks. Package includes RBTE emulation software, installation instructions, user guide and administrator guide. Prerequisite is VPM188DK.
3270BSC188DK	3270 bisync emulator for interactive IBM host access. Emulates a 3271 Model 2 Cluster Control Unit and up to seven devices: 3277/78 Model 2 Displays and one 3287 Model 2 Printer. Compatible with OpenNET for cross-net gateway oriented service. Package includes all emulation software, installation instructions, 3277/78 function key templates, user guide and administrator guide. Prerequisite is VPM188DK.
SXM18848	Eight line communication system extension module hardware. Used when no 188/48 or 188/56 exists in system already. Includes all necessary hardware, cabling and documentation.
SXM354	Additional two line communication system extension module hardware. Used to add 2 async lines to 188 based subsystem via daughter board. Includes all necessary hardware, cabling and documentation. Prerequisite is SBC or SXM 188/48.



3270 SNA COMMUNICATION SUBSYSTEM

- *3270 SNA/SDLC emulator for Intel XENIX* systems and networks*
- *Mainframe data and application access and data download/upload*
- *SNA gateway for OpenNET™ LAN users*
- *3274 cluster controller (PU.T2), 3278 display (LU.T2) and 3287 printer (LU.T2) emulation*
- *Complete menu driven interface and administration delivers ease of use*
- *72 nodes and 16 simultaneous sessions supported*

■ 3270 SNA COMMUNICATION SERVICES

The 3270 Communication Subsystem allows multiple XENIX system users and IBM PC compatibles on an OpenNET network to operate on IBM SNA networks. The subsystem installs in a single non-dedicated XENIX system and runs SNA emulations for system users and OpenNET users. It provides emulation of a 3274 Type 2 Cluster Controller, 3278 Model 2 Displays and 3287 Model 2 Printers. Up to 72 XENIX-NET nodes are supported from a single gateway with 16 simultaneous Logical Unit sessions.

■ SNA COMMUNICATION ENVIRONMENT

This subsystem communicates with the host over dial-up, leased, point-to-point, and multidrop lines, coexisting with IBM equipment. Line speeds up to 9600 bps are supported. The subsystem communicates with a variety of IBM hosts (370, 303X, 308X, 43XX), communication front ends (3705, 3725), access methods (VTAM, TCAM) and applications (CICS, CMS, DSPRINT, ISPF, TSO/SPF). No change is required to the host software for connection and operation of the Intel 3270 SNA Subsystem.

■ OpenNET-SNA CONTROLLER AND DEVICE EMULATION GATEWAY

The 3270 SNA Communication Subsystem can be distributed across an OpenNET network for optimal gateway services. One network node contains the actual SNA communication processor for 3274 Cluster Controller emulation while other nodes have copies of the device emulators. The dispersed device emulators all access the one gateway node for mainframe SNA communication. Up to 16 users can establish and use SNA sessions simultaneously.

■ A SINGLE WORKSTATION FOR ALL PROCESSING AND COMMUNICATION NEEDS

With Intel's 3270 SNA Subsystem, users will no longer require multiple workstations for their various tasks. A single terminal or PC can be used to access local applications and data as well as access mainframe data, applications and reports. Intel's SNA emulator is optimized for very cost effective department automation.

JUNE, 1986
ORDER NUMBER: 270215-001

■ ADVANCED SUBSYSTEM FEATURES

Session hold allows a user to temporarily exit or suspend an SNA session to perform other tasks, while maintaining the host connection, and return to the same connection later.

Multiple gateways can exist on a single OpenNET network in those cases where greater than 16 concurrent sessions are required.

Screen and print capture features allow users to easily log screen data from the current session into any file on the network, and spool printer output to any file. Users can process the captured data files further using standard DOS and XENIX applications.

Printer sharing enables the "local copy" device to be specified as any network printer attached to a node with 3287 Printer Emulation.

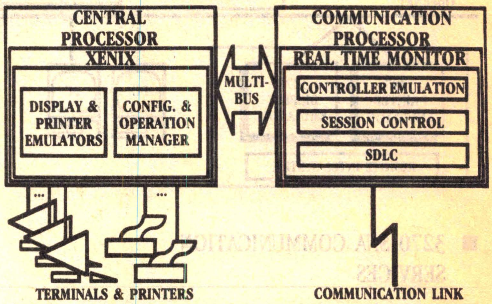
Complete menu-driven user and administration interface reduces installation and maintenance time and enhances user productivity due to low confrontation.

Interactive configuration and terminal definition utilities are included for flexibility in configuring the subsystem for target environments.

For coexistence with Intel's other advanced communication subsystem, VPM188 Async/Bisync Communication Subsystem, assures that a combination of SNA, Bisync and Async lines can be configured and used.

■ HIGH PERFORMANCE MULTI-PROCESSOR SUBSYSTEM ARCHITECTURE

The functions of the 3270 SNA Communications Subsystem are optimally distributed across the system bus. The system's central processor offloads the majority of communication tasks to an advanced communication processor which handles most of the emulation. Printer and display emulation, and configuration and operation administration are done by the central CPU while 3274 SNA Cluster Controller emulation, session control and SDLC are implemented in the communication processor. The net result is higher system performance.



■ A COMPLETE, INTEGRATED SNA COMMUNICATION SOLUTION

The 3270 SNA Communication Subsystem is a fully integrated hardware, firmware and software solution which is ready to install and operate in an Intel XENIX system. Complete installation instructions, administrator's guide and user's guide for both XENIX only and Intel's XENIX enhanced with iBASE are included.

TECHNICAL SPECIFICATIONS

3270 Base Datastream (3270 DSC) SNA Character String (SCS) 1920 Character Device Buffers SNA Communication Protocols SDLC Link Protocols 9600 bps full and half-duplex	SNA PU Type 2 SNA LU Types 1, 2 and 3 Requires synchronous modem and system to modem cabling in addition to leased or dial-up communication line.

ORDERING INFORMATION

3274SNA88	Complete 3270 SNA Communication Subsystem including: iSBC 88/45-based communication controller (double high) with 256K memory and SDLC firmware, 3274 Cluster Controller emulator firmware for single system or OpenNET network SNA gateway operation, 3270 printer support software, 3278 display emulator software, 3287 printer emulator software and complete documentation.

intel

EUROPEAN LITERATURE ORDER FORM

ORDER NUMBER		TITLE	QTY		PRICE	TOTAL
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		
–				X		

PAYMENT

Cheques should be made payable to your local Intel Sales Office (see inside back cover).

Other forms of payment may be available in your country. Please contact the Literature Co-ordinator at your local Intel Sales Office for details.

The Completed form should be marked for the attention of the LITERATURE CO-ORDINATOR and returned to your local Intel Sales Office.

SUB TOTAL ______________

LOCAL TAX ______________

TOTAL ______________

NAME ______________________________

COMPANY ______________________________

ADDRESS ______________________________

____

____

____

PHONE NO. ______________________________

intel

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive
Suite 2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Drive
Suite 214D
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 540-6040

Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
Santa Clara, CA 95051
Tel: (408) 986-8086
TWX: 910-338-0255

COLORADO

Intel Corp.
3300 Mitchell Lane, Suite 210
Boulder 80301
Tel: (303) 442-8088

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

Intel Corp.
650 S. Cherry Street
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06810
Tel: (203) 748-3130
TWX: 710-456-1199

EMC Corp.
222 Summer Street
Stamford 06901
Tel: (203) 327-2934

FLORIDA

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

FLORIDA (Cont'd)

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60172
Tel: (312) 310-8031

INDIANA

Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

Intel Corp.
3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

Intel Corp.
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111

NEW MEXICO

Intel Corp.
8500 Menual Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.*
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391

NORTH CAROLINA

Intel Corp.
5700 Executive Center Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
25700 Science Park Drive
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73116
Tel: (405) 848-8086

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway, Bldg. B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
1513 Cedar Cliff Drive
Camphill 17011
Tel: (717) 737-5035

Intel Corp.*
455 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
400 Penn Center Boulevard
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970

PUERTO RICO

Intel Microprocessor Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-3030

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713) 988-9421

UTAH

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

VIRGINIA

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
155-108 Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Chancellory Park 1
Brookfield 53005
Tel: (414) 784-8087

CANADA

BRITISH COLUMBIA

Intel Semiconductor of Canada, Ltd.
301-2245 W. Broadway
Vancouver V6K 2E4
Tel: (604) 738-6522

ONTARIO

Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TELEX: 06983574

QUEBEC

Intel Semiconductor of Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

*Field Application Location

intel®

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5100
TWX: 910-950-0077

Kierulff Electronics
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
TWX: 910-493-2086

Arrow Electronics, Inc.
1502 Crocker Avenue
Hayward 94544
Tel: (408) 487-4600

Arrow Electronics
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TLX: 888064

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel: (818) 700-6271
TWX: 910-494-2207

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411

Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel: (818) 700-6500

†Hamilton Electro Sales
10950 W. Washington Boulevard
Culver City 90230
Tel: (213) 558-2458
TWX: 910-340-6364

CALIFORNIA (Cont'd)

Hamilton Electro Sales
1361 B West 190th Street
Gardena 90248
Tel: (213) 558-2131

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

Kierulff Electronics
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300

Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

†Kierulff Electronics, Inc.
5650 Jillson Street
Commerce 90040
Tel: (213) 725-0325
TWX: 910-580-3666

Wyle Distribution Group
26560 Agoura Street
Calabasas 91302
Tel: (818) 880-9000
TWX: 818-372-0232

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 843-9953
TWX: 910-595-1572

†Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

Wyle Military
18910 Teller Avenue
Irvine 92750
Tel: (714) 851-9958
TWX: 310-371-9127

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 851-9953
TWX: 910-595-2642

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

CONNECTICUT (Cont'd)

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W., Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer Electronics
221 N. Lake Boulevard
Suite 412
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

†Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway, Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
5835B Peachtree Corners E
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

INDIANA (Cont'd)

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Road #H
Rivers Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350
Twx: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

MTI Systems Sales
13 Fortune Drive
Billerica 01821

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

†Microcomputer System Technical Demonstrator Centers

DOMESTIC DISTRIBUTORS

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03104
Tel: (603) 624-9400

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln East
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 701-734-4388

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Road
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
TWX: 710-541-1560

NEW YORK (Cont'd)

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel: (607) 748-8211
TWX: 510-252-0893

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

Arrow Electronics, Inc.
5240 Greendairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer Electronics
9801 A-Southern Pine Boulevard
Charlotte 28210
Tel: (704) 524-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
TWX: 910-467-8746

Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer Electronics
261 Gibraltar Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Road #190
Houston 77477
Tel: (713) 780-1771
TWX: 910-881-5523

†Pioneer Electronics
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

WISCONSIN (Cont'd)

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 230-3586
TWX: 03-827-642

Hamilton/Avnet Electronics
6845 Rexwood Road Unit 6
Mississauga, Ontario L4V1R2
Tel: (416) 677-0484

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

Hamilton/Avnet Electronics
105-2550 Boundry Road
Burmalay V5M 3Z3
Tel: (604) 272-4242

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
590 Berry Street
Winnipeg R3H OS1
Tel: (204) 775-8661

ONTARIO

Arrow Electronics Inc.
24 Martin Ross Avenue
Downsview M3J 2K9
Tel: (416) 661-0220
TELEX: 06-218213

Arrow Electronics Inc.
148 Colonnade Road
Nepean K2E 7J5
Tel: (613) 226-6903

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

†Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 225-8840
TWX: 06-976-78

QUEBEC

Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TELEX: 05-25596

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
TLX: 05-13388

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

intel

EUROPEAN SALES OFFICES

BELGIUM

Intel Corporation S.A.
Parc Seny
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02) 661 07 11
TELEX: 24814

DENMARK

Intel Denmark A/S*
Glentevej 61 - 3rd Floor
DK-2400 Copenhagen
Tel: (01) 19 80 33
TELEX: 19567

FINLAND

Intel Finland OY
Ruosilantie 2
000390 Helsinki
Tel: (0) 544 644
TELEX: 123 332

FRANCE

Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
Tel: (33) 1 30 57 70 00
TELEX: 69901677

FRANCE (Cont'd)

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842 40 89
TELEX: 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Munchen 2
Tel: (89) 53891
TELEX: 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzerstrasse 75
D-6200 Wiesbaden 1
Tel: (6121) 70 08 74
TELEX: 04168183 INTW D

Intel Semiconductor GmbH
Bruckstrasse 61
7012 Fellbach
Stuttgart
Tel: (711) 58 00 82
TELEX: 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollernstrasse 5*
3000 Hannover 1
Tel: (511) 34 40 81
TELEX: 923625 INTH D

ISRAEL

Intel Semiconductors Ltd*
Atidim Industrial Park
Neve Sharet
Dvora Hanevia
Bldg. No. 13, 4th Floor
P.O. Box 43202
Tel Aviv 61430
Tel: 3-491099/8
TELEX: 371215

ITALY

Intel Corporation Italia Spa*
Milanofiori, Palazzo E/3
20094 Assago (Milano)
Tel: (02) 824 40 71
TELEX: 315183 INTMIL

NETHERLANDS

Intel Semiconductor Nederland B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21 23 77
TELEX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

SPAIN

Intel Iberia
Calle Zurbaran
28-1-IZQDA
28010 Madrid
Tel: (34) 1410 40 04
TELEX: 46880

SWEDEN

Intel Sweden A.B.*
Dalvagen 24
S-171 36 Solna
Tel: 8/7340100
TELEX: 12261

SWITZERLAND

Intel Semiconductor A.G.*
Talackerstrasse 17
8152 Glattbrugg postfach
CH-8065 Zurich
Tel: (01) 829 29 77
TELEX: 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (793) 696 000
TELEX: 444447 INT SWN

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronische Ges.m.b.H.
Meidlinger Hauptstrasse 78
A 1120 Wien
Tel: (222) 83 56 46
TELEX: 11532 BASAT A

Moor Ges.m.b.H.
Storchengasse 1/1/1
A-1150 Wien
Tel: 222-85 86 46

BELGIUM

S.A. Inelco Belgium
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

ITT Electronic Components
rue Colonel Bourgstr. 105a (Bte 3)
B-1140 Brussels
Tel: (02) 735-7125

DENMARK

ITT MultiKomponent A/S
Naverland 29
DK-2600 Gloskrup
Tel: (02) 456 66 45
TX: 33355

FINLAND

Oy Fintronic AB
Melkonkalu 24 A
SF-00210 Helsinki 21
Tel: (0) 692 60 22
TELEX: 124 224 Ftron SF

FRANCE

Generim
Z.A. de Courtaboeuf
Avenue de la Baltique
F-91943 Les Ulis Cedex-B.P. 88
Tel: (1) 69 07 78 78
TELEX: F691700

Jermyn
16, Avenue de Jean-Jaures
94600 Choisy-Le-Roi
Tel: (1) 48 53 12 00
TELEX: 260 967

Metrologie
Tour d' Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel: (1) 47 90 62 40
TELEX: 611-448

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet B.P. 2
92310 Sevre S
Tel: (1) 45 34 75 35
TELEX: 204552

WEST GERMANY

Electronic 2000 Vertriebs A.G.
Stahlgruberring 12
8000 Munich 82
Tel: (89) 42 00 10
TELEX: 522561 EEC D

Jermyn GmbH
Postfach 11 80
Schulstrasse 84
6277 Bad Camberg
Tel: (06434) 231
TELEX: 484426 JERM D

CES Computer Electronics Systems GmbH
AM Moosfeld 37
8000 Munich 82
Tel: (089) 420-430
TELEX: 2180260

Metrologie GmbH
Hansastrasse 15
8000 Munich 21
Tel: (89) 570-940
TELEX: 5213189

Proelectron Vertriebs AG
Max Planck Strasse 1-3
6072 Dreieich
Tel: (6103) 3040
TELEX: 417983

ITT-Multikomponent
Bahnhofstrasse 44
7141 Moeglingen
Tel: (07141) 4870

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (1) 85 62 88
TELEX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: (3) 47 51 51
TELEX: 33638

ITALY

Eledtra 3S S.P.A.
Via G. Watt, 37
I 20143 Milano
Tel: (2) 81 82 1
TELEX: 332332

ITALY (Cont'd)

Intesi
Milanofiori E/5
20090 Assago
Tel: (2) 824701
TELEX: 311351

Lasi Elettronica S.P.A.
V. Le Fulvio Testi, 126
20092 Cinisello Balsamo (Milano)
Tel: 02/2440012
TELEX: 352040 LASIMI I
TELEFAX: 2487717

NETHERLANDS

Koning & HartmanElectrotechniek B.V.
P.O. Box 125
2600 AC Delft
Tel: 15 609-906
TELEX: 31528

NORWAY

Nordisk Elektronic A/S
Post Office Box 130
N-1364 Hvalstad
Tel: (2) 846 210
TELEX: 17546

PORTUGAL

Ditram
Avenida Miguel Bombarda, 133
P-1000 Lisboa
Tel: (1) 154 53 13
TELEX: 14182 Brieks-P

SPAIN

ITT SESA
Miguel Angel 21-3
Madrid 28010
Tel: (1) 419 54 00
TELEX: 27461

Diode
Calle Gandesa 10-4
08028 Barcelona
Tel: (3) 322-12-51
TELEX: 42148

SWEDEN

Nordisk Electronik AB
Huvudstagatan 1
Box 1409
S-171 27 Solna
Tel: (8) 734 97 70
TELEX: 10547

SWITZERLAND

Industrade AG
Hertistrasse
CH-8304 Wallisellen
Tel: (01) 830 50 40
TELEX: 56788 INDEL CH

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House
Jubilee Way
Letchworth
Hertfordshire SG6 1QH
Tel: (0462) 686666
TELEX: 826293

Bytech Ltd.
2 The Western Center
Western Industrial Estate
Bracknell
Berkshire RG12 1RW
Tel: (0344) 482211
TELEX: 848215

Comway Microsystems Ltd.
Market Street
Bracknell
Berkshire
Tel: (344) 55333
TELEX: 847201

IBR Microcomputers Ltd.
Unit 2 Western Center
Western Industrial Estate
Bracknell
Berkshire RG12 1RW
Tel: (0344) 486555
TELEX: 849381

Jermyn Industries Vestry Estate
Oxford Road
Seven Oaks
Kent TN 14 5EU
Tel: (0732) 450144
TELEX: 95142

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: (190) 49307
TELEX: 28817

Rapid Recall, Ltd.
Rapid House/Denmark St.
High Wycombe
Bucks HP11 2ER
Tel: (494) 26 271
TELEX: 837931

IBR
2 The Western Center
Western Road
Bracknell, Berkshire
Tel: (0344) 486-555

intel

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.*
Spectrum Building
200 Pacific Highway
Level 6
Crows Nest, NSW, 2065
Tel: 011-61-2-957-2744
TELEX: 970-20097
FAX: 011-61-2-957-2744

CHINA

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Avenue
Beijing, PRC

HONG KONG

Intel Semiconductor Ltd.*
1701-3 Connaught Centre
1 Connaught Road
Tel: 011-852-5-215-311
TWX: 60410 ITLHK

JAPAN

Intel Japan K.K.
5-6 Tokodai, Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511
TELEX: 03656-160

Intel Japan K.K.*
Komeshin Bldg.
2-1-15 Naka-machi
Atsugi, Kanagawa 243
Tel: 0462-23-3511

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya, Saitama 360
Tel: 0485-24-6871

Intel Japan K.K.*
Ryokuchi-Station Bldg.
2-4-1 Terauchi
Toyonaka, Osaka 560
Tel: 06-863-1091

JAPAN (Cont'd)

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621

Intel Japan K.K.*
Flower-Hill Shin-machi East Bldg.
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel: 03-426-2231

Intel Japan K.K.*
Mitsui-Seimei Musashi-Kosugi Bldg.
915-20 Shinmaruko, Nakahara-ku
Kawasaki-Shi, Kanagawa 211
Tel: 044-733-7011

Intel Japan K.K.
Mishima Tokyo-Kaijo Bldg.
1-1 Shibahon-cho
Mishima-shi
Shizuoka-Ken 411
Tel: 0559-72-4121

KOREA

Intel Semiconductor Asia Ltd.
Singsong Bldg. 8th Floor #906
25-4 Yoido-Dong, Youngdeungpo-Ku
Seoul 150
Tel: 011-82-2-784-8186 or 8286
TELEX: K29312 INTELKO

SINGAPORE

Intel Semiconductor Ltd.
101 Thomson Road
21-06 Goldhill Square
Singapore 1130
Tel: 011-65-250-7811
TWX: RS 39921

TAIWAN

Intel Semiconductor Ltd.
Rm 808, Min Chi Bldg.
746 Min Sheng East Road
Taipei
Tel: 011-886-2-716-9660

*Field Application Location

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

VLC S.R.L. Bartalome Mitre 1711
3 Piso
1037 Buenos Aires
Tel: 011-54-1-49-2092
TELEX: 17575 EDARG-AR

AUSTRALIA

Total Electronics
(Mailing Address)
Private Bag 250
Burwood, Victoria 3125

(Shipping Address)
9 Harker Street
Burwood
Victoria 3125
Tel: 011-61-3-288-4044
TELEX: AA 31261

Total Electronics
P.O. Box 139
Artamon, N.S.W. 2064
Tel: 011-61-02-438-1855
TELEX: 26297

BRAZIL

Elebra Microelectronica S/A
R. Florida, 1821-8 ander
04571 - Sao Paulo-SP
Tel: 011-55-11-533-9977
TELEX: 1125957

CHILE

DIN Instruments
Casilla 6055, Correo 22
Santiago
Tel: 225-8139
TELEX: 440422 Rudy CZ

(Shipping Address)
A102 Greenville Center
3801 Kennett Pike
Wilmington, Delaware 19807

CHINA

Novel Precision Machinery Co., Ltd.
Flat D 20 Kingsford Ind. Bldg.
Phase 1 26 Kwai Hei Street NT
Hong Kong
Tel: 011-852-5-223222
TWX: 39114 JINMI HK

CHINA (Cont'd)

Schmidt & Co. Ltd.
18/F. Great Eagle Centre
Wanchai
Hong Kong
Tel: 011-852-5-822-0222
TWX: 74766 SCHMC HK

HONG KONG

Schmidt & Co. Ltd.
18/F. Great Eagle Centre
Wanchai
Tel: 011-852-5-822-0222
TWX: 74766 SCHMC HK

INDIA

Micronic Devices
65 Arun Complex
D V G Road
Basavan Gudi
Bangalore 560 004
Tel: 011-91-812-600-631
TELEX: 011-5947 MDEV

Micronic Devices
104/109C Nirmal Industrial Estate
Sion (E)
Bombay 400 022
Tel: 011-91-22-48-61-70
TELEX: 011-71447 MDEV IN

Micronic Devices
R-694 New Rajinder Nager
New Delhi 110 060

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. Room 407
2-14-1 Asano, Kokurakita-Ku
Kitakyushu City 802
Tel: (093) 511-6471
TELEX: AECKY 7126-16

C. Itoh Micronics Corp.
OS 85 Bldg. 2-6-5 Suda-Cho
Kanda Chiyoda-Ku, Tokyo 101
Tel: (03) 256-2211
TELEX: (03) 252-3774

JAPAN (Cont'd)

Ryoyo Electric Corporation
Shuwa Sakurabashi Bldg.
4-5-4 Hatchobori
Chuo-Ku, Tokyo 104
Tel: (03) 555-4811

Tokyo Electron Ltd.
Shinjuku Nomura Bldg.
1-26-2 Nishi-Shinjuku
Shinjuku-Ku, Tokyo 160
Tel: (03) 343-4411
TELEX: 232-2220 LABTEL J

KOREA

J-TEK Corporation
2nd Floor, Government Pension Bldg.
24-3 Yoido-Dong
Youngdungpo-Ku
Seoul 150
Tel: 011-82-2-782-8039
TELEX: KODIGIT K25299

Samsung
23rd Fl. Dong Bang Bldg.
1502-KA Taepyung-RU
Chung-Ku
Seoul
Tel: 777-78
TELEX 27970 KORSST K

MEXICO

DICOPEL S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 9011525561321
TELEX: 1773790 DICOME

NEW ZEALAND

Northrup Instruments & Systems Ltd.
459 Kyber Pass Road
P.O. Box 9464, Newmarket
Auckland 1
Tel: 011-64-9-501-219, 501-801, 587-037
TELEX: NZ21570 THERMAL

Northrup Instruments & Systems Ltd.
P.O. Box 2406
Wellington 856658
TELEX: NZ3380

PAKISTAN

Computer Applications Ltd.
7D Gizri Boulevard
Defence
Karachi-46
Tel: 011-92-21-530-306
TELEX: 24434 GAFAR PK

Horizon Training Co., Inc. (Agent)
1 Lafayette Center
1120 20th Street N.W.
Suite 530
Washington, D.C. 20036
Tel: (202) 887-1900
TWX: 248890 HORN

SINGAPORE

General Engineers Corporation Pty. Ltd.
203 Henderson Road
1102 Henderson Industrial Park 0315
Tel: 011065-271-3163
TELEX: RS23987 GENERCO

SOUTH AFRICA

Electronic Building Elements, Pty. Ltd.
(Mailing Address)
P.O. Box 4609
Pretoria 0001
Tel: 011-27-12-469921
TELEX: 3-22786 SA

(Shipping Address)
Pine Square, 18th Street
Hazelwood Pretoria

TAIWAN

Mitac Corporation
No. 585 Ming Sheng E. Road
Taipei
Tel: 011-96-2-501-8231
TELEX: 11942 TAIAUTO

VENEZUELA

P. Benavides CA
Arilanes a Rio
Resdencias Kamarata
Local 4 a LZ
Caracas
Tel: (582) 571-0396

*Field Application Location

CG-11/6/86

DOMESTIC SERVICE OFFICES

ALABAMA

Intel Corp.
5015 Bradford Drive, #2
Huntsville 35805
Tel: (205) 830-4010

ARIZONA

Intel Corp.
11225 N. 28th Dr. #D214
Phoenix 85029
Tel: (602) 869-4980

Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

ARKANSAS

Intel Corp.
P.O. Box 206
Ulm 72170
Tel: (501) 241-3264

CALIFORNIA

Intel Corp.
21515 Vanowen
Suite 116
Canoga Park 91303
Tel: (818) 704-8500

Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: 1-800-468-3548

Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143

Intel Corp.
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8211
TWX: 910-339-9279
910-338-0255

Intel Corp.
2000 E. 4th Street
Suite 110
Santa Ana 92705
Tel: 1-800-468-3548
TWX: 910-595-2475

Intel Corp.
2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1740

Intel Corp.
4350 Executive Drive
Suite 150
San Diego 92121
Tel: (619) 452-5880

COLORADO

Intel Corp.
650 South Cherry
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130

FLORIDA

Intel Corp.
6363 N.W. 6th Way
Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

GEORGIA

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 441-1171

ILLINOIS

Intel Corp.
300 N. Martingale Rd.
Suite 300
Schaumburg 60194
Tel: (312) 310-8034

INDIANA

Intel Corp.
8777 Purdue Rd., #125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

KENTUCKY

Intel Corp.
3525 Tatescreek Road, #51
Lexington 40502
Tel: (606) 272-6745

MARYLAND

Intel Corp.
4th Floor Product Service
7833 Walker Drive
Greenbelt 20770
Tel: (301) 220-3313

MASSACHUSETTS

Intel Corp.
3 Carlisle Road
Westford 01886
Tel: (617) 692-1060

MICHIGAN

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8905

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

NEW JERSEY

Intel Corp.
385 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0820
TWX: 710-991-8593

Intel Corp.
Raritan Plaza III
Raritan Center
Edison 08817
Tel: (201) 225-3000

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

OHIO

Intel Corp.
Chagrin-Brainard Bldg.
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

Intel Corp.
6500 Poe
Dayton 45414
Tel: (513) 890-5350

OREGON

Intel Corp.
10700 S.W. Beaverton-Hillsdale Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel: (313) 354-1540

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628
TWX: 910-874-1347

Intel Corp.
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-2820
TWX: 910-860-5617

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

VIRGINIA

Intel Corp.
1603 Santa Rosa Rd., #109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: 1-800-468-3548
TWX: 910-443-3002

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Brookfield 53005
Tel: (414) 784-8087

CANADA

Intel Corp.
190 Attwell Drive, Suite 103
Rexdale, Ontario
Canada M9W 6H8
Tel: (416) 675-2105

Intel Corp.
620 St. Jean Blvd.
Pointe Claire, Quebec
Canada H9R 3K2
Tel: (514) 694-9130

Intel Corp.
2650 Queensview Drive, #250
Ottawa, Ontario,
Canada K2B 8H6
Tel: (613) 829-9714

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700

ILLINOIS

300 N. Martingale, #300
Schaumburg 60173
Tel: (312) 310-5700

MASSACHUSETTS

3 Carlisle Road
Westford 01886
Tel: (617) 692-1000

MARYLAND

7833 Walker Dr., 4th Floor
Greenbelt 20770
Tel: (301) 220-3380

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5–6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel Paris
1 Rue Edison, BP 303
78054 Saint-Quentin en Yvelines
France

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

HONG KONG
Intel Semiconductor Ltd.
1701 Connaught Centre
1 Connaught Road
Hong Kong

Printed in U.S.A. 0187/60K/RRD PM

ISBN 1-55512-064-4